ALGORITHMIC EDUCATION THEORY

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Timothy Priestley Novikoff

January 2013

ALGORITHMIC EDUCATION THEORY

Timothy Priestley Novikoff, Ph.D.

Cornell University 2013

In order to build toward an algorithmic theory of education we construct simple, idealized mathematical models of students, learning, educational software and educational software usage data. The models are created by taking concepts from the psychology literature and commercial educational software, stripping them down to bare mathematical essentials, and then rigorously analyzing these models.

We consider the *spacing effect* from the psychology literature and model the notion of *spaced repetition* as simple constraints on mathematical sequences. Though the constraints are simply stated – that each occurrence of any element in the sequence fall within a given interval of possible distances beyond the previous occurrence – the mathematical problems that arise from these constraints are subtle. We present novel mathematical techniques suited to these problems.

We also consider educational software usage data, and consider the task of measuring the amount of educational content a student must have mastered at any given time given that they produced some specific usage data. We find that once properly defined, the task is again subtle and requires carefully constructed algorithms, which in turn require careful mathematical analysis.

Finally we consider the notion that it is easier for students to learn new concepts that are related to already-familiar concepts, and we present a novel network optimization problem inspired by this notion.

BIOGRAPHICAL SKETCH

Tim Novikoff was born and raised New York City, attending public schools from kindergarden through twelfth grade, working occasionally as an actor while attending LaGuardia High School for Music and Art and Performing Arts. He attended NYU from 1998 to 2003, where he majored in mathematics and theater. From 2003 to 2007 he taught at Stuyvesant High School as a member of the New York City Teaching Fellows program while working towards his Masters in Secondary Mathematics Education at City College of New York at night. He began graduate school at Cornell University in 2007. While in graduate school his research interests ranged from studying the mathematics of slime molds to the mathematics of educational software algorithms and data mining, while his hobbies included dancing, cooking, poker, iPhone app development and entrepreneurship. He leaves graduate school as the CEO of Vantageous Inc., and as a lecturer teaching Beginning iPhone App Development in the Computer Science Department at Cornell University.

Dedicated to those who dedicate their lives to teaching.

ACKNOWLEDGEMENTS

This dissertation is the fruit of a years-long collaboration with Steve Strogatz and Jon Kleinberg. None of it would have been possible – nor would I be the same person I am today – without their lovingly given encouragement, their wisdom and guidance in research direction, and their carefully shared technical feedback. There would be nothing here without them. Similarly, their encouragement, direction and detailed feedback with respect to writing and communication have not only been critical to my ability to communicate the material contained herein, but have molded me personally. The transformational impact that Steve and Jon had on my life cannot be overstated and will never be forgotten.

I'd also like to thank Richard Rand, Paul Ginsparg, Bobby Kleinberg and other members of the Cornell faculty who have shared with me so much during my years in Ithaca.

I also owe a debt of gratitude to my peers in the Center for Applied Math, and in particular to Diarmuid Cahalane, Joel Nishimura and Johan Ugander. I would never have achieved much without the countless technical conversations I shared with them throughout my time in Ithaca. Whether at a white board in CAM, coffee at Gimme or CTB, dinner at my apartment on Buffalo Street, or pints at the Chapter House, the conversations were always insightful and always enjoyable.

It takes a village, and I could not list all the peers who helped me sort out my thoughts or shared in my adventure. Bruno Abrahao, Sam Arbesman, Adam Chacone, Phil Meerkamp and Josh Schwartz come to mind, but there are too many to list here. Thanks to everybody who helped and supported me throughout my time at Cornell.

I'd also like acknowledge the helpful conversations I've had with friends and peers outside of graduate school. I'm lucky to have so many mathematically minded friends who can understand what I do at a detailed technical level, and looking forward to sharing my research with them is part of what has driven me all this time. Again, there are too many to list here, but certainly Boris Granovskiy, Joel Lewis, Joseph Stern, and Jan Siwanowicz come to mind.

Finally, I could not have gotten through any of this without the unwavering love and support of Oana Pascu, from the day I applied to graduate school to the day I graduated.

Thank you all so very much.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Great teachers use colloquialisms that seem to reveal something about how they think about their students. The students are said to be *solidifying their understanding*, or sometimes letting things *sink in*; students *get it* but sometimes they seem to *have it and then lose it*; they are *making the connections* or even *connecting the dots*; eventually things start to *click*. These expressions reveal the teachers' mental models of students and of learning.

Perhaps great teachers are great in part because they continuously update their mental models of their students based on feedback that they get from them, and then adapt their teaching methods to this evolving mental model. This suggests a way in which mathematical modeling can inform the algorithms that drive educational software, so that the software could adapt to users the way great teachers adapt to their students.

Algorithms driving educational software could be informed by mathematical models of students, learning and educational subject matter. The software could fit a parametrized mathematical model of the users to the incoming stream of data generated by them. This way, the algorithm driving the software could continuously adapt to the user, much the way great teachers seem to adapt their methods to the continuously changing mental model they keep of their students. The mathematical models could be designed so that the parameters combine to represent the same sort of things that teachers talk about.

Perhaps one parameter could represent the understanding that the user has of a particular bit of educational content, so that an increase in the parameter

represents the student's *solidifying understanding* of that bit of educational content – their increasing ability to recall a fact, for example, or their ability to recall or reconstruct the proof of a theorem.

Perhaps modeling of the educational content itself could be aided by mathematical structures as well. A lesson comparing two bits of subject matter – *connecting the dots* for the student – could be modeled by an edge in a network, where nodes represent mathematical or historical facts, for example, and edges represent conceptual connections between these facts.

Perhaps the language that teachers use – of students *getting it*, or of students *having it and then losing it*, or of things *clicking* – suggest that binary variables that flip over time can be a productive way to model a student's mastery of bits of subject matter.

This dissertation is an exploration of these ideas. Although the ideas above are not new, the literature has lacked the kind of analyses of simple mathematical models that typically serve as foundations for the intuitions of engineers. This dissertation is a small step toward the goal of developing an algorithmic theory of education based on mathematical models of students, educational content and learning.

The title is a nod to the subject of *algorithmic game theory*, which is a well-developed theory based on highly idealized mathematical models. Although the models are idealized, the intuitions gained by studying algorithmic game theory are useful for software engineers in the now-giant industry of online ad auction design.

With the current boom in internet-enabled educational software gaining

speed, the time is ripe to develop an analogous theory which can serve as a foundation for the intuitions of software engineers working in the growing industry of educational software algorithm design.

Chapters 2 and 3 of this dissertation, adapted from *Education of a Model Student* [**?**], present idealized models of students and the educational process, and examine some combinatorial issues that arise out of attempting to sequence educational content in a way that aligns with the principles of the *spacing effect* from the psychology literature. (The spacing effect explains the intuitive notion that as one solidifies one's understanding or knowledge of educational content, one gains the ability to retain the understanding or knowledge for longer and longer periods of time.)

Chapter 4 considers the task of analyzing usage data from educational software. Building on idealized models of mastery and forgetting – models wherein mastery of educational content is modeled using binary variables that flip over time – and educational software usage data, a method for estimating the user's total mastery over time is presented. In addition, a framework for reasoning about the effectiveness of the estimation method is presented as well.

Chapter 5 introduces a novel networks optimization problem inspired by the notion that concepts are related, and that it is easier to learn a concept in a familiar domain than an unfamiliar domain; is is easier to absorb concepts if one can connect the new concepts to familiar ones.

The last 10 years have seen an boom in so-called *biologically inspired* mechanical engineering. If the foundations are laid by computer scientists and applied mathematicians now, the next 10 years may see a boom in *teacher inspired* soft-

ware engineering, and the author of this dissertation certainly hopes this will be the case.

CHAPTER 2

EDUCATION OF A MODEL STUDENT

A dilemma faced by teachers, and increasingly by designers of educational software, is the tradeoff between teaching new material and reviewing what has already been taught. Complicating matters, review is useful only if it is neither too soon nor too late. Moreover, different students need to review at different rates. We present a mathematical model that captures these issues in idealized form. The student's needs are modeled as constraints on the schedule according to which educational material and review are spaced over time. Our results include algorithms to construct schedules that adhere to various spacing constraints, and bounds on the rate at which new material can be introduced under these schedules.

2.1 Motivation

In his 2009 speech to the National Academy of Sciences, President Barack Obama exhorted the audience to imagine some of the things that could be made possible by a commitment to scientific research, including the invention of "learning software as effective as a private tutor"[**?**]. This chapter is a modest step in that direction.

An important general challenge for the design of educational software is to fully incorporate the results of empirical research on how people learn. Such research endeavors to provide principles for how to choose what is taught, how to present it, and how to sequence the material. Ultimately, educational software will require mechanisms for managing the constraints that arise when the more

well-established of these principles are applied in different settings.

Here we develop and analyze an idealized mathematical model for incorporating a fundamental class of such constraints into educational software — constraints arising from the importance of timing and review in the presentation of new material. For example, software for building vocabulary must determine when to introduce new words which the student has not yet learned, and when to review words whose definitions the student has successfully recalled in the past. The issue is similar to that faced by a high school math teacher deciding how often to schedule lessons that involve trigonometry, or by a piano teacher who needs to decide how much time a student should devote to practicing scales while also learning to play new pieces.

The study of the importance of timing with respect to review dates to at least 1885 [**?**]. The notion that it is better to spread studying over time instead of doing it all at once is called the *spacing effect*. In 1988 Dempster noted that "the spacing effect is one of the most studied phenomena in the 100-year history of learning research" [**?**]. As Balota et al. point out in their review [**?**], "spacing effects occur across domains (e.g., learning perceptual motor tasks vs. learning lists of words), across species (e.g., rats, pigeons, and humans), across age groups and individuals with different memory impairments, and across retention intervals of seconds to months." See Refs. [**?**] and [**?**] for reviews. For further results and background, see Refs. [**?**], [**?**] and [**?**]. Work on exploiting the spacing effect to build vocabulary (in humans) includes influential scholarship by Bjork [**?**], as well as systems developed and studied by Pimsleur [**?**] and Wozniak [**?**], [**?**].

Review is an important part of the learning process, but the extent to which

review is needed varies by student. Some students need to review early and often, while others can learn a lot before going back to review at all. Personalized educational software of the future will likely need to fit a model to the student using the software and then schedule review in a way that is tailored to the model.

With this in mind, we envision a system in which the software designer can specify a schedule for the introduction of new material, together with a schedule by which the review of existing material is spaced over time. What we find, however, is that the resulting scheduling problems are mathematically subtle: existing techniques do not handle scheduling problems with spacing constraints of this type.

Our main contribution is to develop an approach for reasoning about the feasibility of schedules under spacing constraints. We begin by introducing a stylized mathematical model for the constraints that arise from the spacing effect, and then consider the design of schedules that incorporate these constraints.

2.2 Models

Roughly speaking, we model the introduction and review of material as a sequence of abstract educational units, and we model the needs of the student using two sequences, $\{a_k\}$ and $\{b_k\}$: after an educational unit has been introduced, the "ideal" time for the student to see it for the $(k+1)^{\text{st}}$ time is between a_k and b_k time steps after seeing it for the k^{th} time. We also model various educational outcomes that the designer of educational software may seek to achieve. The motivation and technical details are given below.

2.2.1 The Educational Process

We imagine the underlying educational software as implementing a process that presents a sequence of abstract educational units which could represent facts such as the definitions of vocabulary words, concepts such as trigonometric identities, or skills such as playing a scale on the piano. For example, the sequence $u_1, u_2, u_3, u_1, u_4, \ldots$, indicates that educational unit u_1 was introduced at the first time step and reviewed at the fourth time step. This sequence is the *schedule* according to which the units will be presented. We also allow our schedules to contain *blanks*, or time steps in which no educational unit is presented; thus, an arbitrary schedule will have each entry equal to either an educational unit or a blank.

This is a highly idealized model. It ignores possible relationships between units, such as the etymological (and potentially pedagogically useful) connection between the vocabulary words "neophyte" and "neologism", for example, or the dependence of trigonometry on more basic concepts in geometry. It also treats all units as equal. Thus it does not capture, for example, that an experienced pianist may benefit more from practicing a scale than practicing Twinkle Twinkle Little Star.

The real-life educational process is nuanced, complex, and context-dependent. Future work in building models for educational software may introduce more complexity to this model, or simply reduce scope and model more specific situations. Here, in the interest of generality as well as mathematical tractability, we use this very simple model of the general educational process in order to create a formalism that captures spacing constraints and their role in reviewing material after it has been introduced.

2.2.2 Spacing Constraints

A large body of empirical work in learning research has studied the *expanding* nature of optimal review. For example, when students first learn a new vocabulary word, they must review it soon or else they likely lose the ability to correctly recall the definition. If they do review it before forgetting it, they will then generally be able to go longer than before without needing review. By repeatedly reviewing, the student "builds up" the ability to go longer and longer without seeing the word while maintaining the ability to recall its definition. However, reviewing a word too soon after studying it can reduce the benefit of the review. These are the principles of the well-established theory of *expanded retrieval*; see [**?**] for a review specifically on expanded retrieval. More generally, see Refs. [**?**] – [**?**].

We want a simple formalism that captures the need to review an educational unit on a schedule that "expands" the spacing between successive viewings. We wish to leave the exact rate of expansion under the control of the software designer, motivated by the goal of creating different schedules for different students. We thus imagine that the designer of the software specifies a set of *spacing constraints*, consisting of an infinite sequence of ordered pairs $(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k), \ldots$, where $a_k \leq b_k$ are positive integers for all k. Intuitively, the idea is that for each educational unit u_i in the schedule, the designer wants the gap in the schedule between the k^{th} and $(k+1)^{\text{st}}$ occurrences of u_i to have length in $[a_k, b_k]$. The fact that a student can go longer between occurrences as they gain familiarity with the educational unit is represented by the assumption that the numbers a_k and b_k are weakly increasing: we impose the requirement that $a_k \leq a_{k+1}$ and $b_k \leq b_{k+1}$ for all k.

Thus, our key definition is that a schedule *satisfies* a set of spacing constraints if for each u_i in the schedule, the $(k+1)^{\text{st}}$ occurrence of u_i in the schedule comes between a_k and b_k positions (inclusive) after the k^{th} occurrence. Roughly speaking, the numbers b_k model how long the student can retain learned material. The numbers a_k model how long a student should wait before review is beneficial. This captures the notion that there is an ideal time to review. If review is done too early, it is not very beneficial. If it is done too late, then the student will forget the material in the interim.

We should emphasize that this is, by design, a very simple model of spacing constraints and their role in learning. A more nuanced model might allow for "blurry" boundaries for the intervals $[a_k, b_k]$, in which there is a numerical penalty for missing the interval by a small amount. Another refinement would be to allow the model to discriminate between educational units, since in real life some things are easier to learn than others. However, the simple model described above captures the essence of the phenomenon we are investigating, and will allow for mathematical analysis that helps us understand the mechanics of scheduling review in an optimal way.

2.2.3 Educational Goals

We consider two natural goals for the designer of the educational software. The mission of the software could be to teach students in such a way that they grow their knowledge without bound, never forgetting anything along the way; a sort of "lifelong learning" approach to education. Alternatively, the mission of the software could simply be to get students familiar with a certain set of educa-

tional units by a particular point in time, regardless of whether they are destined to forget what they learned quickly thereafter; something like the studying technique known as "cramming". We address both goals in this chapter.

We model the first goal by saying that a schedule exhibits *infinite perfect learning* with respect to some spacing constraints if (i) it satisfies the spacing constraints, and (ii) it contains infinitely many educational units, each of which occurs infinitely often. Thus if the constraints represented the needs of a student, then with such a schedule the student would, over the course of the infinite sequence, learn an infinite set of educational units without ever forgetting anything.

For the second goal we consider a finite sequence, representing the presentation of material up to a test or performance. For a positive integer n, we say that the sequence is a *cramming* sequence, and exhibits *bounded learning* of *order* n, if (i) it satisfies the spacing constraints, (ii) it contains at least n distinct educational units such that, if the unit occurs a total of k times in the sequence, then its last occurrence is within b_k positions of the end of the sequence. Condition (ii) captures the requirement that the student should still be able to recall all of the n educational units at the end of the sequence, which was the whole point of "cramming". Note that although the spacing constraints have been respected up to the end of the finite sequence, there is no guarantee that the sequence could be extended while continuing to satisfy the spacing constraints.

With respect to infinite perfect learning, we will be interested in the rate at which the student would learn if taught according to the schedule. To this end, we define *the introduction time function*: for a given schedule of educational units, let t_n denote the position in the schedule of the first occurrence of the n^{th} distinct

educational unit. Thus the slower the growth of t_n, the faster new educational units are being introduced.

We will be considering questions of the following nature. Given spacing constraints $\{(a_k, b_k)\}$, does there exist a schedule that exhibits infinite perfect learning, or bounded learning, with respect to the constraints? If so, how can we construct such schedules? And when such a schedule exists, what rate of learning (as measured by the sequence $t_1, t_2, \ldots$) is achievable? These are fundamental problems that would be faced by an educational software designer seeking to incorporate spacing constraints in the design of the underlying algorithms. As we will see, despite the simply stated formulation of these questions, the combinatorial challenges that they lead to quickly become quite intricate.

2.3 Results

The spacing constraints are described by two infinite sequences of parameters, $\{a_k\}$ and $\{b_k\}$. In this section we describe how choices for these parameters affect the rate at which new educational units can be introduced into the schedule, and we describe how schedules can be tailored to particular parameter regimes and educational goals.

2.3.1 Overview of Results

We begin by examining the first main issue of this chapter: the trade-off between (i) the rate at which b_k grows as a function of k and (ii) the rate at which t_n grows as a function of n. Informally, if b_k grows relatively slowly, then a lot of

time must be spent on review rather than on introducing new units, and hence t_n must grow more quickly, corresponding to slower learning. It is clear that $t_n \geq n$ for any schedule, since even without review we can only introduce one educational unit per time step. With this in mind, we consider the following pair of questions that explore the two sides of the trade-off. First, is there a set of spacing constraints for which t_n is close to this trivial bound, growing nearly linearly in n? Second, as we require b_k to grow slower as a function of k, it becomes more difficult to achieve infinite perfect learning. Is there a set of spacing constraints for which infinite perfect learning is possible, and for which b_k grows as a polynomial function of k? We answer both of these questions in the affirmative.

In *The Recap Method* we describe a set of spacing constraints for which a schedule can be constructed that exhibits infinite perfect learning, and where the rate of learning is relatively quick. In the schedule we construct, t_n grows as $\Theta(n \log n)$, and in fact

$$t_n \leq n \cdot (\lfloor \log_2 n \rfloor + 1).$$

Recalling that we must have $t_n \geq n$ for all n for any schedule, we see that the recap schedule achieves infinite perfect learning with only a modest increase on this bound — that is, with a relatively small amount of review. In the Supporting Information (Chapter 3) we show how to construct a schedule where t_n grows at a rate that, in some sense, can be as close to linear as desired. In *Superlinearity of the Introduction Time Function*, we show that there can be no schedule whatsoever such that t_n grows as $O(n)$.

In *The Slow Flashcard Schedule*, we show a set of spacing constraints for which a_k and b_k grow polynomially in k and for which infinite perfect learning is possi-

ble. With these spacing constraints based on much smaller a_k and b_k, the schedule we construct has a slower rate of learning; we show that t_n is bounded below by $\Omega(n^2)$ and bounded above by $O(n^3)$, in contrast to the schedule from *The Recap Method* for which the introduction time function grows as $\Theta(n \log n)$. The gap between the quadratic lower bound and the cubic upper bound is an interesting open question; we give numerical evidence that in fact the lower bound may be tight, and that t_n grows as $O(n^2)$. Much of this is done in the Supporting Information (Chapter 3).

Following these results, we turn to the second main issue in this chapter, which is to identify general possibility and impossibility results for satisfying classes of spacing constraints. We first show, in *Flexible Students*, that the difficulty in achieving infinite perfect learning stems, in a sense, from the fact that the numbers a_k are growing: specifically, we show that for *any* spacing constraints in which $a_k = 1$ and $b_k \to \infty$, it is possible to construct a schedule that exhibits infinite perfect learning. The construction introduced here demonstrates a general method that can be adapted to many sets of spacing constraints with $a_k > 1$ as well.

Thus far, we have only considered spacing constraints that allow for the construction of schedules that exhibit infinite perfect learning. In *The Finicky Slow Student*, we show that there exist spacing constraints for which no schedule can exhibit infinite perfect learning. In particular, we build this impossibility result from an extreme case where $a_k = b_k = f(k)$ and $f(k)$ is a function that grows slowly in k. This represents a setting in which the student insists on reviewing material on an extremely precise and plodding schedule, with no room for error.

The difficulty in constructing a schedule for such a set of constraints is that as

the knowledge base — the number of educational units introduced — grows, so does the need to review, and so the potential for scheduling conflicts increases. The *slower* the student (the slower the growth of $f(k)$), the fewer educational units can be put on the back burner, so to speak, while the student focuses on new units. The more *finicky* the student (the smaller the *windows* $b_k - a_k$), the less wiggle room there is in scheduling review.

All of this matches intuition. Students who don't need to review much and aren't too picky about when the review needs to happen can be taught a lot, and fast. But students who need a lot of review and who only derive benefit from very well-timed review will be more difficult to teach. The educational mantra is "every child can learn", but designers of personalized educational software may find that scheduling the educational process for some students is, at the least, more difficult for some than for others.

In *Cramming*, we show that every student can cram. More precisely, we show that for any set of spacing constraints and any n, it is possible to construct a finite sequence that exhibits bounded learning of order n. Consistent with the discussion that accompanied the definition of bounded learning earlier, the construction assures nothing about whether the sequence can be extended beyond this moment of "expertise" at its end.

Finally, also in *Cramming*, we explore the question of how much can be crammed in a given amount of time. Given a set of spacing constraints and a finite number T, we derive a non-trivial upper bound on those n for which bounded learning of order n is possible using only T time steps while adhering to the given spacing constraints.

2.3.2 The Recap Method

Here we explore spacing constraints that allow for infinite perfect learning with a rapid learning rate — that is, where the introduction time function, t_n, grows slowly.

Consider the spacing constraints

$$a_k = 2^k$$

and

$$b_k = 2^{k-1}(k+1).$$

A schedule that allows for infinite perfect learning with respect to these spacing constraints can be described as follows. To find the first $(k+1) \cdot 2^k$ entries of the schedule, consider a depth-first postorder traversal of a full binary tree of depth k with 2^k leaves labeled $u_1, u_2, \ldots, u_{2^k}$ from left to right. (A depth-first postorder traversal of a tree is a particular order for visiting the nodes of a tree, defined as follows. Starting at the root v of the tree, the depth-first postorder traversal is first applied recursively to each subtree below v one at a time; *after* all these traversals are done, then the root v is declared to be visited.) Begin with an empty sequence. Every time a leaf is visited, append the sequence with the corresponding educational unit. Every time a non-leaf node is visited, append the sequence with the units corresponding to all of the descendant leaves, in left-to-right order. The resulting sequence gives the first $(k+1) \cdot 2^k$ entries in the schedule.

Thus, using $k = 2$ we have that the first 12 entries of the schedule are

$$u_1, u_2, u_1, u_2, u_3, u_4, u_3, u_4, u_1, u_2, u_3, u_4.$$

We call this *the recap schedule* because it incorporates periodic review of everything that has been learned so far, like a teacher saying "okay, let's recap".

In this schedule, the number of time steps between the k^{th} and $(k+1)^{\text{st}}$ occurrence of any particular unit is always between 2^k and $2^{k-1}(k+1)$, with both bounds actually achieved for each k. Since infinitely many units are seen infinitely often, due to the properties of depth-first traversals, this establishes that the schedule allows for infinite perfect learning with respect to the given spacing constraints. Calculations which establish these facts are shown in the Supporting Information (Chapter 3).

Further calculations, also shown in the Supporting Information (Chapter 3), establish that t_n grows as $\Theta(n \log n)$ in this schedule. More precisely,

$$\frac{1}{2} \cdot n \cdot (\lfloor \log_2 n \rfloor + 1) \leq t_n \leq n \cdot (\lfloor \log_2 n \rfloor + 1).$$

By generalizing the construction of the schedule, using a more general class of trees, we can show that for a large class of functions $r(n)$, schedules can be constructed that exhibit infinite perfect learning for which t_n grows as $\Theta(n \cdot r^{-1}(n))$. The class includes functions $r(n)$ that grow arbitrarily fast, and so in that sense we can create schedules for which t_n grows at a rate that is as close to linear as desired. The downside of these schedules is that they require increasingly lax spacing constraints as the growth rate of t_n approaches linearity: the schedules that we construct for which t_n grows as $\Theta(n \cdot r^{-1}(n))$ require b_k, as well as $b_k - a_k$, to grow as $\Theta(k \cdot r(k))$. Relevant calculations and proofs for this, too, can be found in the Supporting Information (Chapter 3).

2.3.3 Superlinearity of the Introduction Time Function

Although our constructions are able to achieve introduction times t_n that grow arbitrarily close to linearly in n, we can also show that an actual linear rate of growth is not achievable: for schedules that exhibit infinite perfect learning, the introduction time function t_n must be superlinear. More precisely, we show that for any schedule that exhibits infinite perfect learning with respect to any spacing constraints $\{(a_k, b_k)\}$, there cannot be a constant c such that $t_n \leq cn$ for all n.

To prove this, we consider an arbitrary set of spacing constraints $\{(a_k, b_k)\}$ and an arbitrary schedule that exhibits infinite perfect learning with respect to these constraints, and assume for the sake of contradiction that there were a constant c such that $t_n \leq cn$ for all n. Let

$$\hat{b}_k = \sum_{j=1}^{k} b_j$$

and let n_0 be any integer such that $n_0 > \hat{b}_{c+1}$. By our assumption, at least n_0 educational units have been introduced by the time step cn_0. In general, for any schedule that exhibits infinite perfect learning, any unit that has been introduced by time step t will have occurred k times by time step $t + \hat{b}_k$, by the definition of $\hat{b}_k$. Thus at least n_0 units will have occurred $c + 1$ times by time $cn_0 + \hat{b}_{c+1}$. So

$$cn_0 + \hat{b}_{c+1} \geq (c + 1)n_0$$

since each time step corresponds to at most one educational unit.

Subtracting cn_0 from both sides we have that $\hat{b}_{c+1} \geq n_0$, which contradicts our choice of n_0.

2.3.4 The Slow Flashcard Schedule

One can describe the construction of the recap schedule without using depth-first traversals or full binary trees, but rather using a deck of flashcards. Doing so will shed some light on the recap schedule, and will also serve as a useful jumping-off point for discussing the very different schedule which is the focus of this section. So we begin our discussion here by revisiting the recap schedule.

Imagine a deck of flashcards, with the k^{th} card corresponding to an educational unit u_k. Thus the top card corresponds to u_1, the next card corresponds to u_2, etc. Then we construct a schedule as follows. At every step, we present to the student the educational unit corresponding to the top card, and then reinsert the card into position 2^k, where k is the number of times we have presented the educational unit corresponding to that card, up to and including this latest time step. Thus first we present u_1, then we remove it from the deck and reinsert it into position 2 in the deck. Then we present u_2, then remove it and reinsert it into position 2 in the deck. Then u_1 is again on top and so we present it for a 2^{nd} time and then remove it and reinsert it into position $2^2 = 4$. This process produces the recap schedule.

In this section we consider a schedule that is much more difficult to describe explicitly than the recap schedule, but whose construction can similarly be described in terms of a deck of flashcards. Instead of reinserting into position 2^k as above, though, we reinsert into position $k + 1$. Carefully applying this rule shows the first few entries of the schedule to be

$$u_1, u_2, u_1, u_2, u_3, u_1, u_3, u_2, u_4, u_3.$$

Of all schedules that can be constructed with a similar flashcard-reinsertion

scheme using some strictly increasing reinsertion function $r(k)$ with $r(1) > 1$, it is this schedule, constructed using $r(k) = k + 1$, which progresses through the deck the slowest. For this reason, we call this *the slow flashcard schedule*.

In the Supporting Information (Chapter 3) we show that the slow flashcard schedule exhibits infinite perfect learning with respect to the spacing constraints with $(a_k, b_k) = (k, k^2)$. Thus the slow flashcard schedule provides a dramatic alternative to the recap schedule. Whereas b_k and $b_k - a_k$ both grew exponentially in k in the recap schedule, here they grow quadratically and yet still allow for infinite perfect learning.

Numerical simulations shown in the Supporting Information (Chapter 3) suggest that this schedule in fact exhibits infinite perfect learning even with respect to the much tighter spacing constraints with $(a_k, b_k) = (k, 2k)$. This would make the contrast with the recap schedule even more stark.

The tradeoff for this slow growth in b_k is the speed at which the knowledge base grows. Whereas t_n, the time needed for the knowledge base to achieve size n, grew as $\Theta(n \log n)$ in the recap schedule, here it is bounded below by $\Omega(n^2)$. A proof of this can be found in the Supporting Information (Chapter 3), along with numerical evidence that it in fact grows as $\Theta(n^2)$.

The spacing constraints in *The Recap Method* and *The Slow Flashcard Schedule* are tailored to allow for existence proofs that certain bounds on t_n, b_k and $b_k - a_k$ can be achieved in the context of infinite perfect learning. The methods used to describe the schedules, though, suggest general principles for how to construct schedules with desirable properties. Moreover, we note that the schedules constructed are relevant to any set of spacing constraints that are more relaxed than

the given ones: if a schedule exhibits infinite perfect learning with respect to spacing constraints $\{(a_k, b_k)\}$, then it also exhibits infinite perfect learning with respect to $\{(a'_k, b'_k)\}$ when $a'_k \leq a_k$ and $b'_k \geq b_k$ for all k.

In the next section, *Flexible Students*, we begin with a general class of students and build schedules tailored to each individual student in the class.

2.3.5 Flexible Students

What if the student didn't need to wait at all in order to derive benefit from studying? In other words, what if $a_k = 1$ for all k? This allows for a technique in constructing educational processes that we call *hold-build*: sequencing the educational units that are known to the student in a "holding pattern" so that they meet the spacing constraints, while showing new educational units in quick repetition (thereby "building" them up). The only assumptions that are needed for the construction, besides $a_k \equiv 1$, are that $b_1 \geq 2$ and that $b_k \to \infty$. (Note that b_k is already required to be weakly increasing.)

We define the sequence HB_m to be the infinite sequence that starts with u_m, contains u_m in every other entry, and cycles through units $u_1, \ldots, u_{m-1}$ in the remaining entries. So, for example,

$$\begin{aligned} HB_2 &= u_2, u_1, u_2, u_1, u_2, u_1, \ldots \\ HB_3 &= u_3, u_1, u_3, u_2, u_3, u_1, u_3, u_2, u_3, u_1, \ldots \\ HB_4 &= u_4, u_1, u_4, u_2, u_4, u_3, u_4, u_1, u_4, u_2, u_4, \ldots \end{aligned}$$

Now, consider an arbitrary set of spacing constraints such that $a_k \equiv 1$, $b_1 \geq 2$, and $b_k \to \infty$. For ease of discussion, we say that an educational unit is at *level k* when it has been shown exactly k times. We construct the educational process that assures infinite perfect learning as follows.

First, present unit u_1. Then present units according to HB_2. So far so good; so long as units are presented according to HB_2, we know the spacing constraints will be met, since $b_k \geq 2$ for all k. Meanwhile, the levels for u_1 and u_2 can be built up without bound.

Continue HB_2 as long as necessary until it is feasible to move on to HB_3. In other words, show enough of HB_2 so that if the educational process up to that point were followed by an indefinite run of HB_3, then the spacing constraints would be met. This is guaranteed to be true after a finite number of time steps since $b_k \to \infty$ and HB_2 is periodic.

Thus, once enough of HB_2 has been shown, we show as much of HB_3 as necessary until we can afford to move on to HB_4. Then we show as much of HB_4 as necessary until we can afford to move on to HB_5, and we continue building the educational process like this indefinitely.

The educational process formed by concatenating hold-build patterns in this way assures infinite perfect learning, and it applies to any set of spacing constraints with $a_k \equiv 1$, $b_1 \geq 2$ and $b_k \to \infty$. Thus we actually have a class of spacing constraints where infinite perfect learning is possible and yet b_k can grow *arbitrarily slowly*. The trade-off for an exceedingly slow-growing b_k will again be a fast-growing t_n, corresponding to a slow rate of learning. The exact rate will depend on the exact rate of growth of b_k.

To give a concrete example of this hold-build construction and an accompanying calculation of t_n, we can consider the simple case where $b_k = k + 1$. (Note that since we only require that b_k be *weakly* increasing, this is by no means the slowest-growing choice for b_k.) Then, by carrying out the construction above, we have that the sequence is

$$u_1, \underbrace{u_2, u_1, u_2, u_1, u_2}_{HB_2}, \underbrace{u_3, u_1, u_3, u_2, u_3, u_1, u_3, u_2, u_3}_{HB_3}, \underbrace{u_4, u_1, \ldots}_{HB_4}$$

with $4k - 3$ time steps spent in HB_k for every k. Thus, since u_i will be introduced one time step after finishing the HB_{i-1} part of the schedule, we have that

$$t_n = 1 + \Big(\sum_{k=2}^{n-1} (4k - 3) \Big) + 1 = 2n^2 - 5n + 4.$$

The idea behind the hold-build construction, namely the method of putting some units in a "holding pattern" while others are being "built up", could readily be used to construct schedules assuring infinite perfect learning for many sets of spacing constraints with $a_k > 1$ as well. (Or spacing constraints with $b_1 = 1$, for that matter.) It is a tool that can be used to tailor educational processes to model students in general.

2.3.6 The Finicky Slow Student

We now give a set of spacing constraints $\{(a_k, b_k)\}$ for which no schedule can exhibit infinite perfect learning. In other words, for any schedule that contains infinitely many educational units that each appear infinitely often, the schedule cannot satisfy $\{(a_k, b_k)\}$.

Our spacing constraints are defined simply by $a_k = b_k = k$. We call this set

of constraints *the finicky slow student* because $b_k - a_k$ is so small, and because b_k grows so slowly with k.

Suppose, for the sake of contradiction, that there were a schedule that exhibited infinite perfect learning with respect to this set of spacing constraints. We say that the schedule *incorporates* educational unit u_i if the unit is presented infinitely many times, and all occurrences of u_i satisfy the spacing constraints. Given the structure of the spacing constraints, this means that if unit u_i is incorporated, and if it is first presented at step τ, then it must be presented at exactly the steps $\tau, \tau + 1, \tau + 3, \tau + 6, \tau + 10, \ldots$

We can assume without loss of generality that the first educational unit, u_1, is incorporated and that its first appearance is at step $\tau = 0$. (Letting time start at 0 here allows for cleaner calculations.) Then we know that u_1 is presented at steps $\tau_1, \tau_2, \tau_3, \ldots$ where

$$\tau_i = \sum_{k=1}^{i} a_k.$$

Now we just need to show that no other unit can be incorporated without creating a scheduling conflict — in other words, without needing to eventually be scheduled at a step of the form τ_i.

Suppose another unit, call it u_2, were incorporated, with its first appearance at step s_0. Then we know that u_2 must appear at precisely the steps in the sequence $s_1, s_2, s_3, \ldots$ where

$$s_i = s_0 + \sum_{k=1}^{i} a_k.$$

We show that there must be some step common to both sequences $\{s_i\}$ and $\{\tau_i\}$, which will be our contradiction, since at most one educational unit can be shown in each entry of the schedule.

We begin by noting that $s_i - \tau_i = s_0$ for all i, and that $s_{i+1} - s_i = \tau_{i+1} - \tau_i = a_i$ for all i. Now choose k large enough so that $\tau_{k+1} - \tau_k > s_0$. Then

$$\tau_{k+1} > \tau_k + s_0 = s_k.$$

Thus for sufficiently large k, we have that $\tau_{k+1} > s_k$. Now let m be the smallest number such that $\tau_{m+1} > s_m$. We know $m \geq 2$, since $\tau_1 = 0$ and $\tau_2 = 1$ by construction. We claim that $\tau_m = s_{m-1}$.

If $\tau_m > s_{m-1}$ then m would not be the smallest number such that $\tau_{m+1} > s_m$ (since then $m - 1$ would also qualify), so $\tau_m \not> s_{m-1}$.

If $\tau_m < s_{m-1}$, then we have that

$$\tau_m < s_{m-1} < s_m < \tau_{m+1}.$$

This implies that

$$s_m - s_{m-1} \leq \tau_{m+1} - \tau_m - 2,$$

since all s_i and τ_i are integer-valued. Thus

$$a_{m-1} \leq a_m - 2,$$

so

$$a_m - a_{m-1} \geq 2,$$

which is not possible since $a_k - a_{k-1} = 1$ for all k. So $\tau_m \not< s_{m-1}$.

Thus we have that $\tau_m = s_{m-1}$. This cannot be, of course, since only one educational unit can be presented at any given time step. So it must be that the finicky slow student cannot incorporate more than one unit.

This proof holds not only for the spacing constraints $a_k = b_k = k + 1$, but for any spacing constraints such that $a_k = b_k = f(k)$ where $f(k)$ is any integer

sequence such that $f(k+1) - f(k) \in \{0, 1\}$ and $f(k) \to \infty$. The exact choice doesn't matter; the finickiness ($a_k = b_k$) and the slowness ($f(k+1) - f(k) \in \{0, 1\}$) are sufficient to carry out the proof as written, but with the final argument using $a_k - a_{k-1} \leq 1$ instead of $a_k - a_{k-1} = 1$.

2.3.7 Cramming

The focus up until now has been on infinite perfect learning, but there could be less ambitious goals for a student. We turn our attention now to cramming. At the end of this section, we address the question of how much cramming can be done in a given amount of time. We begin here with a positive result, showing that for every positive integer n and every set of spacing constraints with $b_k \to \infty$, there exists a sequence that achieves bounded learning of order n.

We consider an arbitrary set of spacing constraints and proceed by induction on n. It is clear that bounded learning is possible for $n = 1$; the sequence consisting simply of u_1 satisfies the definition.

Now suppose that bounded learning of order n is possible, and let S_n be a sequence of length T_n that achieves this. We now construct a new sequence, S_{n+1} of length T_{n+1}, that achieves bounded learning of order $n+1$.

Recall from *Flexible Students* that the *level* of an educational unit at time t in a sequence is the number of times it has appeared prior to time t. The basic idea behind the construction of S_{n+1} is to start by building up the level of u_1 until it is at a level m such that $b_m > T_n$. Then we use the next T_n steps to present units u_2, $u_3, \ldots, u_{n+1}$ according to the sequence S_n. When that is done, the time limit of b_m

has still not been reached for u_1, and hence the sequence satisfies the definition of bounded learning of order $n + 1$.

Formally, let m be the smallest integer such that $b_m > T_n$. Then present unit u_1 at time $t = 0$ and at times

$$t = \sum_{i=1}^{j} a_i$$

for $j = 1, 2, \ldots, m - 1$. Present a blank in the sequence at every other time step in between. Then, starting at time

$$t = 1 + \sum_{i=1}^{m-1} a_i,$$

present units $u_2, u_3, \ldots, u_{n+1}$ according to the T_n elements of the sequence S_n. This sequence, through time

$$T_{n+1} = T_n + \sum_{i=1}^{m-1} a_i,$$

is our new sequence S_{n+1}. By construction it satisfies the conditions of bounded learning of order $n + 1$. By induction, then, bounded learning of order n is possible for all positive integers n. Since our choice of spacing constraints with $b_k \to \infty$ was arbitrary, this establishes that bounded learning of order n is possible for all n for any set of spacing constraints with $b_k \to \infty$.

In the construction above, it is entirely possible that one or more units would begin to violate the spacing constraints even one time step later. Little is assured other than the educational units having met the scheduling constraints up to a certain time step. We call this sort of construction "cramming" because it presents the material with a particular target time in view and without regard to the scheduling of material after this target time, like a student cramming for a final exam who doesn't worry about how much will be retained after the test.

Condition (ii) of our definition of bounded learning models the notion of

studying up to a point in time and then being able to remember everything that was studied for at least one more time step, as if there were a quiz lasting one time step which would occur in the time step immediately following cramming sequence. We could similarly model the notion of a quiz that lasts m time steps by requiring that if a unit's last occurrence is s time steps from the end of the sequence, and the unit occurs a total of k times in the sequence, then $s+m$ must be less than or equal to b_k (instead of simply s.) We note that our results regarding cramming sequences could be adapted to such an alternative model.

We turn now to the issue of how much can be crammed in a given amount of time. Given a set of spacing constraints $\{(a_k, b_k)\}$, and a positive integer T, we can put an upper bound on the numbers n for which bounded learning of order n is possible in T time steps. If we let $m(i)$ denote the smallest integer k such that $b_k \geq i$, then it can be shown that n must satisfy

$$\sum_{i=1}^{n} m(i) \leq T$$

and

$$\Big(\sum_{j=1}^{m(n)-1} a_j \Big) + n \leq T.$$

Each inequality implicitly bounds n from above. These bounds reflect the constraints imposed by the limited number of time steps available, and the notions that $\{a_k\}$ and $\{b_k\}$ represent limitations on how fast the level of an educational unit can be built up and how long educational units can be remembered. Details can be found in the Supporting Information (Chapter 3).

Despite the negative connotations associated with cramming, the basic idea can actually be useful in a number of settings in real life. A traveler may only care to learn a language enough to travel in a foreign country just once, for example, or a performer may only need to have a certain skill set on the day

of a performance and not necessarily after that. Perhaps educational software of the future will have tunable parameters that allow the student (or teacher or parent) to set the goal of the educational process. This way the software may not only adapt to the natural abilities of the students, but also to their personal goals.

2.4 Future Work

The possibilities for future work seem limitless. A more complete theory of infinite perfect learning could be one goal. This would include more techniques for constructing educational processes tailored to students, and a more complete theory relating a_k and b_k to the maximum rate at which the model student can accrue knowledge.

A major goal should be a truly adaptive educational process: one that adapts to the student in real time. For example, in this chapter we model the educational process as a sequence designed to satisfy a set of constraints fixed in advance, but an alternate approach would be to test the student's knowledge throughout the process, and for the schedule to be controlled by an on-line algorithm that chooses the next unit based on the answers the student has given. This would model the process of a teacher observing student progress before deciding what to teach next.

Modeling this situation would be an exciting challenge. The interaction between the two on-line algorithms, one modeling the student and the other modeling the teacher, promises to be complex and fascinating, and hopefully enlightening and useful to future designers and engineers of educational software.

There is much opportunity here for mathematical modeling, theoretical calculations and numerical simulations that shed light on what makes an effective teacher and how educational software can adapt in real time to user behavior.

Another area for future work is the design and analysis of models that are tailored to specific subjects. Perhaps a model involving a *network* of educational units could be used to investigate the phenomenon that it is often easier to learn a set of facts that somehow "reinforce" each other than a set of unrelated facts. Introducing relationships between educational units calls for new models of the student's reception of the units, which in turn call for novel educational processes.

Yet another avenue of research is empirical work. The techniques and intuitions gained from theoretical work should be put to use to create actual educational software. Then data from real students can be collected and the process of using the data to validate and refine the models can begin.

Finally, the mathematics of managing spacing constraints in sequences could find additional applications beyond those considered above, for example, to task management in parallel processing or the study of multi-tasking in humans.

2.5 Conclusion

The models presented in this chapter are simple and theoretical. Designers of educational software will likely need to implement models and algorithms that are more complex, and tailored to the educational content being delivered. It is

our hope that work on simple theoretical models will provide the foundations of intuition for designers of educational software, in much the same way that algorithmic game theory does for engineers who work in online ad auctions and other related fields.

With the current boom in educational software — not to mention the humanoid robot teacher industry [**?**] — it is clear that the time has come to develop a theory of algorithmic education.

CHAPTER 3

SUPPORTING INFORMATION FOR EDUCATION OF A MODEL STUDENT

This is the Supporting Information *Education of a Model Student* (Chapter 2). It is organized into three sections, *The Recap Method*, *The Slow Flashcard Schedule* and *Cramming*. Each section contains the proofs and derivations referenced from the section of the same name in *Education of a Model Student* (Chapter 2).

3.1 The Recap Method

In *The Recap Method* in *Education of a Model Student* (Chapter 2), we described a schedule in terms of a depth-first traversal of a full binary tree, and claimed that it conformed to the spacing constraints

$$\begin{aligned} a_k &= 2^k \\ b_k &= 2^{k-1}(k+1). \end{aligned}$$

We also claimed that for this schedule, which we refer to as *the recap schedule*, the number of time steps before n distinct educational units have been introduced, denoted in *Education of a Model Student* (Chapter 2) as t_n, grows as $\Theta(n \log n)$, and that for a certain class of functions $r(n)$ we can explicitly construct schedules for which t_n grows as $\Theta(n \cdot r^{-1}(n))$.

First we prove the results about the original recap schedule in *The Recap Schedule*, and then we generalize these results in *The Generalized Recap Schedule*. In both subsections we use notation slightly different from *Education of a Model Student* (Chapter 2) by subtracting 1 from all the indices of the educational units,

so that the lowest index is 0. This will make calculations easier, and will allow for a cleaner generalization later on.

3.1.1 The Recap Schedule

We begin by reiterating how to construct the recap schedule. To find the first $(k+1)2^k$ entries of the schedule, consider a depth-first postorder traversal of a full binary tree of height k with 2^k leaves labeled $u_0, u_1, \ldots, u_{2^k-1}$ from left to right. (See Figure 1.) Begin with an empty sequence. Every time a leaf is visited, append the sequence with the corresponding educational unit. Every time a non-leaf node is visited (after both children have been visited), append the sequence with the units corresponding to all of the descendant leaves, in left-to-right order. To be clear, we mean for the leaves to have height 0, their parents to have height 1, etc.

Thus, using $k = 2$ we have that the first 12 entries of the schedule are

$$u_0, u_1, u_0, u_1, u_2, u_3, u_2, u_3, u_0, u_1, u_2, u_3.$$

It should be noted that, by the properties of depth-first postorder traversal, this description defines a *unique* sequence, since the first $(k+1)2^k$ elements of the sequence are the same regardless of whether one considers a tree of height k or one of height greater than k. Thus in the discussion and proofs below we simply assume that the tree being discussed is always of sufficient height to include all of the relevant nodes.

The following lemma, which should be clear from the diagram in Figure 1, is justified by the fact that a depth-first postorder traversal of a tree will visit, in

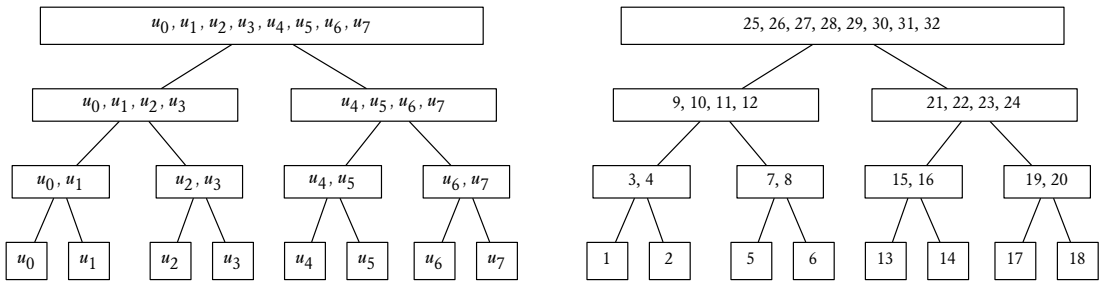

Figure 3.1: The full binary tree on the left has each node labeled with the corresponding educational units in the construction of the recap schedule. The tree on the right is identical, except the nodes are labeled with the corresponding time steps. The corresponding schedule, up to and including the left-most node at height $k = 2$, is $u_0, u_1, u_0, u_1, u_2, u_3, u_2, u_3, u_0, u_1, u_2, u_3, \ldots$.

order of increasing height, each node on the path from any given leaf to the root. The lemma follows from this and from basic properties of a full binary tree.

Lemma 1 (The Recap Lemma) *In the construction of the recap schedule, the left-most node at height k corresponds to the* $(k+1)^{\text{st}}$ *occurrences of units* u_0, u_1,..., u_{2^k-1}, *and the sibling of that node corresponds to the* $(k+1)^{\text{st}}$ *occurrences of units* u_{2^k}, ..., $u_{2^{k+1}-1}$.

To make the discussion below more concise, we introduce some notation. Let $T_i(k)$ be the index of the k^{th} occurrence of unit u_i in the sequence. Note that $t_n = T_n(1)$ by this definition. Thus the recap lemma states that in the recap schedule, the left-most node at height k corresponds to $T_i(k+1)$ for $i \in [0, 2^k - 1]$, and the sibling of that node corresponds to $T_i(k+1)$ for $i \in [2^k, 2^{k+1} - 1]$.

We are now ready to prove the statements about the recap schedule from *Education of a Model Student* (Chapter 2).

Theorem 1 (Asymptotics of the Introduction Time Function) *In the recap schedule,* $t_n = T_n(1)$ *grows as* $\Theta(n \log n)$.

By the recap lemma and properties of depth-first postorder traversal, at time step $T_{2^k}(1)$ units $u_0, u_1, \ldots, u_{2^k-1}$ have each occurred exactly $k+1$ times, and nothing else has occurred at all. Therefore,

$$T_{2^k}(1) = 2^k \cdot (k+1),$$

and so

$$T_n(1) = n \cdot (\log_2 n + 1)$$

for n of the form $n = 2^k$. This establishes that $T_n(1)$ grows as $\Theta(n \log n)$ when considered as a function of integers of the form $n = 2^k$.

Since $T_n(1)$ increases monotonically in n, this implies that $T_n(1)$ grows as $\Theta(n \log n)$ when considered as a function of all positive integers.

Theorem 2 (Bounds on the Introduction Time Function) *In the recap schedule,*

$$T_n(1) \leq n \cdot (\lfloor \log_2 n \rfloor + 1)$$

and

$$\frac{1}{2} \cdot n \cdot (\lfloor \log_2 n \rfloor + 1) \leq T_n(1)$$

for all n.

In general, by time step $T_n(1)$ only units $u_0, u_1, \ldots, u_{n-1}$ have already occurred at all, by the properties of depth-first postorder traversal, and each at most $\lfloor \log_2 n \rfloor + 1$ times, by the recap lemma. Therefore,

$$T_n(1) \leq n \cdot (\lfloor \log_2 n \rfloor + 1).$$

Furthermore, by time step $T_n(1)$ all units with index less than $\frac{1}{2}n$ have occurred exactly $\lfloor \log_2 n \rfloor + 1$ times, again by the properties of depth-first postorder traversal and the recap lemma. Therefore,

$$\frac{1}{2} \cdot n \cdot (\lfloor \log_2 n \rfloor + 1) \leq T_n(1).$$

Theorem 3 (Adherence to Spacing Constraints) *The recap schedule adheres to the spacing constraints*

$$\begin{aligned} a_k &= 2^k \\ b_k &= 2^{k-1}(k+1). \end{aligned}$$

Our goal is to show that

$$a_k \leq T_i(k+1) - T_i(k) \leq b_k,$$

for all i, k. We will show this by calculating the minimum and maximum possible values of $T_i(k+1) - T_i(k)$.

Since the tree is a full binary tree, we have that for any k all the subtrees with roots at height k are identical. Therefore all values of $T_i(k+1) - T_i(k)$ which occur in the context of any given subtree at height k must occur in the context of the subtree rooted at the left-most node at height k. Thus, for any k we only need to consider $i < 2^k$ in order to find the minimum and maximum values of

$$T_i(k+1) - T_i(k).$$

By construction

$$T_j(k+1) - T_i(k+1) = j - i$$

whenever $i < j < 2^k$. Also, since $T_i(k)$ is monotonic in i,

$$T_j(k) - T_i(k) \geq j - i$$

whenever $i < j$. Therefore, for all i and j such that $i < j < 2^k$ we have that

$$T_j(k+1) - T_j(k) \leq T_i(k+1) - T_i(k).$$

Thus, the maximum value of $T_i(k+1) - T_i(k)$ must occur when $i = 0$ and the minimum value must occur when $i = 2^k - 1$.

Thus if we can show that

$$T_{2^k-1}(k+1) - T_{2^k-1}(k) \geq 2^k$$

and that

$$T_0(k+1) - T_0(k) \leq 2^{k-1}(k+1)$$

for all k then we will be done. We will in fact show that we have equality in both cases; we will show that

$$T_{2^k-1}(k+1) - T_{2^k-1}(k) = 2^k$$

and

$$T_0(k+1) - T_0(k) = 2^{k-1}(k+1)$$

for all k.

By construction, the last entry of the schedule due to the left-most node at height k corresponds to $T_{2^k-1}(k+1)$, and the last entry of the schedule due to the right child of that node corresponds to $T_{2^k-1}(k)$. Since in depth-first postorder traversal each node is visited immediately after its right child, we have that

$$T_{2^k-1}(k+1) - T_{2^k-1}(k) = 2^k,$$

corresponding to the 2^k entries of the schedule due to the left-most node at height k.

Meanwhile, $T_0(k+1)$ and $T_0(k)$ refer to the first entry of the schedule due to the left-most nodes at heights k and $k-1$ respectively. Thus $T_0(k+1) - T_0(k)$ will be equal to the number of entries in the schedule due to the left-most node at height $k-1$, plus the number of entries due to the subtree whose root is the right sibling of the left-most node at height $k-1$.

The first quantity is 2^{k-1}, by construction. As for the second quantity, since the subtree in question corresponds to k occurrences of 2^{k-1} units, we have that the second quantity is equal to $k \cdot 2^{k-1}$. Thus

$$\begin{aligned} T_0(k+1) - T_0(k) &= 2^{k-1} + k \cdot 2^{k-1} \\ &= 2^{k-1}(k+1). \end{aligned}$$

It should be noted that in the recap schedule, not only is the gap between the k^{th} and $(k+1)^{\text{st}}$ occurrence of any unit always between a_k and b_k, it is always in fact *equal* to one or the other. This does not hold true in the analogous result for the generalized recap schedule, detailed in the next section. The statement and proof in this section are worded such that they generalize most cleanly in the next section.

Corollary 1 (Window Growth) *The minimal window length required for the recap schedule, $b_k - a_k$, grows as $\Theta(k \cdot 2^k)$*

This follows from the bounds above, since

$$b_k - a_k = 2^{k-1}(k+1) - 2^k$$

$$
\begin{aligned}
&= 2^{k-1}(k+1-2) \\
&= \frac{1}{2} \cdot 2^k \cdot (k-1).
\end{aligned}
$$

3.1.2 The Generalized Recap Schedule

We now move on to generalizing these results by considering a class of schedules which we call the *generalized recap schedule*. To that end, we consider a class of trees more general than the full binary tree; we consider trees where at any given height all of the nodes have the same number of children, but where this number is not necessarily 2 for every height (as it is in a full binary tree).

To construct a generalized recap schedule, begin with any sequence of positive integers

$$\{q(i)\},$$

such that $q(i) \geq 2$ for all i. Then define a sequence

$$\{r(i)\}$$

by setting $r(0) = 1$ and letting

$$r(i) = \prod_{j=1}^{i} q(j)$$

for $i \geq 1$.

Now, to find the first $(k+1)r(k)$ entries of the schedule, consider a depth-first postorder traversal of a tree of height k with $r(k)$ leaves labeled u_0, u_1,..., $u_{r(k)-1}$ from left to right, and such that all the nodes at height j have exactly $q(j)$ children. Begin with an empty sequence. As before, every time a leaf is visited, append the sequence with the corresponding educational unit. Every time a

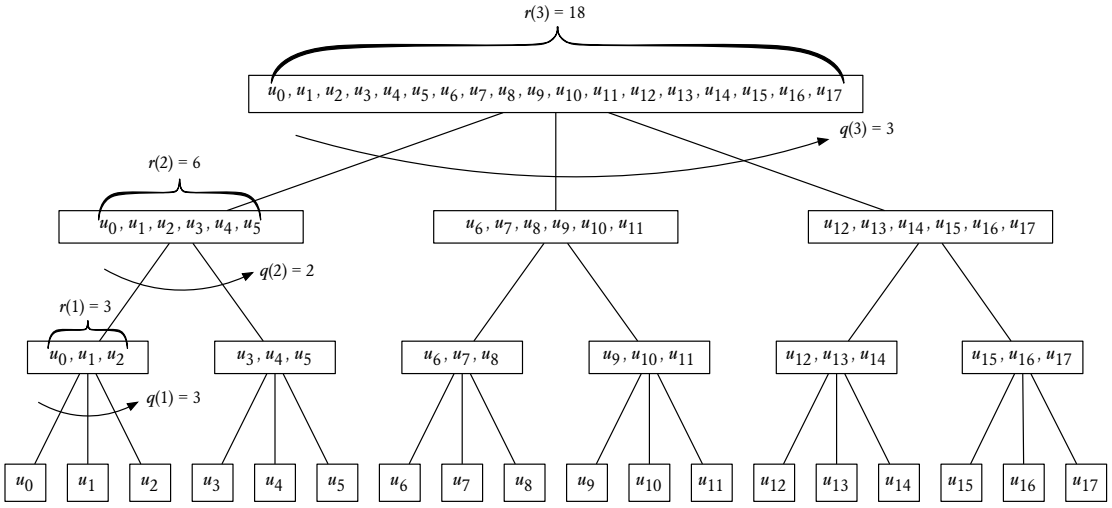

Figure 3.2: A tree made using $q(1) = 3$, $q(2) = 2$ and $q(3) = 3$, with each node labeled with the corresponding educational units in the construction of the generalized recap schedule. The corresponding schedule, up to and including the left-most node at height $k = 2$, is $u_0, u_1, u_2, u_0, u_1, u_2, u_3, u_4, u_5, u_3, u_4, u_5, u_0, u_1, u_2, u_3, u_4, u_5 \ldots$.

non-leaf node is visited (after all of its children have been visited), append the sequence with the units corresponding to all of the descendant leaves, in left-to-right order. Again, we mean for the leaves to have height 0, their parents to have height 1, etc.

Thus, for example, using $q(i) \equiv 2$ we simply have the original recap schedule, whereas using

$$\{q(i)\} = 3, 2, \ldots$$

as in the diagram in Figure 2 we have that the first 18 entries of the schedule are

$$u_0, u_1, u_2, u_0, u_1, u_2,$$

$$u_3, u_4, u_5, u_3, u_4, u_5,$$

$$u_0, u_1, u_2, u_3, u_4, u_5.$$

Again it should be noted that, by the properties of depth-first postorder traversal, this description defines a *unique* sequence, since the first $(k+1)r(k)$ elements of the sequence are the same regardless of whether one considers a tree of height k or one of height greater than k. Thus in the discussion and proofs below we simply assume that the tree being discussed is always of sufficient height to include all the relevant nodes.

We would like to extend r^{-1} so that it has an inverse defined for all positive integers. Where r^{-1} is not naturally defined (i.e. for positive integers n such that $r(i) \neq n$ for any i), we define $r^{-1}(n)$ to simply be $r^{-1}(m)$ where m is the largest number less than n such that $r(i) = m$ for some i. (Thus, for example, if $q(k) \equiv 2$, then we have that $r(k) = 2^k$ and $r^{-1}(n) = \lfloor \log_2 n \rfloor$.)

We note that $r^{-1}(n) + 1$ can be interpreted as the height of the lowest ancestor of leaf u_n that is the left-most node at that height. Thus, by the properties of depth-first postorder traversal, when leaf u_n is visited only nodes of height less than or equal to $r^{-1}(n)$ have already been visited.

The generalization of the recap lemma is evident from the diagram in Figure 2.

Lemma 2 (The Recap Lemma – Generalized) *The left-most node at height k corresponds to the* $(k+1)^{\text{st}}$ *occurrences of units*

$$u_0, u_1, \ldots, u_{r(k)-1},$$

and in general the j^{th} *node at height k, counting from left to right, corresponds to the* $(k+1)^{\text{st}}$ *occurrences of units*

$$u_{(j-1)r(k)}, \ldots, u_{jr(k)-1}.$$

In other words, the left-most node at height k corresponds to $T_i(k+1)$ *for* $i \in [0, r(k)-1]$, *and the* j^{th} *node at height k corresponds to* $T_i(k+1)$ *for*

$$i \in [(j-1)r(k), jr(k)-1].$$

With this in hand, we prove the main results about the generalized recap schedule. The structure of all of the proofs mirrors the structure of analogous proofs in *The Recap Schedule*.

Theorem 4 (Asymptotics of the Introduction Time Function – Generalized) *In the generalized recap schedule,* $T_n(1)$ *grows as* $\Theta(n \cdot r^{-1}(n))$.

By the properties of depth-first postorder traversal and the recap lemma, at time step $T_{r(k)}(1)$, units u_0, u_1,..., $u_{r(k)-1}$ have each occurred exactly $k+1$ times, and nothing else has occurred at all. Therefore,

$$T_{r(k)}(1) = r(k) \cdot (k+1),$$

and so

$$T_n(1) = n \cdot (r^{-1}(n)+1)$$

for n of the form $n = r(k)$ for some positive integer k. This establishes that $T_n(1)$ grows as $\Theta(n \cdot r^{-1}(n))$ when considered as a function over integers of the form $n = r(k)$.

Since $T_n(1)$ increases monotonically in n, this implies that $T_n(1)$ grows as $\Theta(n \cdot r^{-1}(n))$ when considered as a function of all positive integers, so long as $(n+1) \cdot r^{-1}(n+1)$ grows as $\Theta(n \cdot r^{-1}(n))$. This last statement is true since $r^{-1}(n)$ grows at most logarithmically (since, by construction, $r(k) \geq 2^k$ for all k), and so we are done.

Theorem 5 (Bounds on the Introduction Time Function – Generalized) *In the generalized recap schedule,*

$$T_n(1) \leq n \cdot (r^{-1}(n) + 1)$$

and

$$\frac{1}{2} \cdot n \cdot (r^{-1}(n) + 1) \leq T_n(1)$$

for all n.

In general, by time step $T_n(1)$ only units $u_0, u_1, \ldots, u_{n-1}$ have already occurred at all, by the properties of depth-first postorder traversal, and each at most $r^{-1}(n) + 1$ times. Therefore,

$$T_n(1) \leq n \cdot (r^{-1}(n) + 1).$$

Furthermore, by time step $T_n(1)$ all units with index less than $\frac{1}{2}n$ have occurred exactly $r^{-1}(n) + 1$ times. To see why this is true, consider an arbitrary n and let j represent the left-to-right index of the ancestor of leaf u_n that is at height $r^{-1}(n)$. (Thus, if the ancestor of leaf u_n at height $r^{-1}(n)$ is immediately to the right of the left-most node at that height then $j = 2$, whereas if it is the right-most sibling of the left-most node at that height, then $j = q(r^{-1}(n) + 1)$. Note that $j \geq 2$ since, as noted earlier, $r^{-1}(n) + 1$ is the height of the lowest ancestor of u_n that is the left-most node at that height.)

By the properties of depth-first postorder traversal, when leaf u_n is visited, all $j - 1$ nodes at height $r^{-1}(n)$ to the left of the ancestor of u_n at that height will have been visited already, as will all of the descendants of these $j - 1$ nodes. Such leaves will have indices 0 through

$$(j - 1) \cdot r(r^{-1}(n)) - 1.$$

(Note that by construction, $r(r^{-1}(n)$ is not generally equal to n, but rather to the greatest number m less than n such that $r(k) = m$ for some k.) Thus, at $T_n(1)$, we have that all units with index less than

$$(j-1) \cdot r(r^{-1}(n))$$

have been seen $r^{-1}(n) + 1$ times. Since

$$n < j \cdot r(r^{-1}(n))$$

and $j \geq 2$, this establishes that at least $\frac{1}{2} \cdot n$ units have been seen at least $r^{-1}(n) + 1$ times by $T_n(1)$. Thus

$$\frac{1}{2} \cdot n \cdot (r^{-1}(n) + 1) \leq T_n(1).$$

Theorem 6 (Adherence to Spacing Constraints – Generalized) *The recap schedule adheres to the spacing constraints*

$$\begin{aligned} a_k &= r(k) \\ b_k &= r(k) \cdot k - r(k-1) \cdot (k-1) \end{aligned}$$

Our goal is to show that

$$r(k) \leq T_i(k+1) - T_i(k) \leq r(k) \cdot k - r(k-1) \cdot (k-1),$$

for all i, k. We will show this by calculating the minimum and maximum possible values of $T_i(k+1) - T_i(k)$.

For any k, all the subtrees with roots at height k are identical except for a shift in the labels on the leaves, by construction. Therefore all values of $T_i(k+1) - T_i(k)$ which occur in the context of any given subtree at height k must occur in the context of the subtree rooted at the left-most node at height k. This subtree

corresponds to units $u_0, \ldots, u_{r(k)}$. Thus, for any k we only need to consider $i < r(k)$ in order to find the minimum and maximum values of

$$T_i(k+1) - T_i(k).$$

By construction

$$T_j(k+1) - T_i(k+1) = j - i$$

whenever $i < j < r(k)$. Also, since $T_i(k)$ is monotonic in i,

$$T_j(k) - T_i(k) \geq j - i$$

whenever $i < j$. Therefore, for all i and j such that $i < j < r(k)$ we have that

$$T_j(k+1) - T_j(k) \leq T_i(k+1) - T_i(k).$$

Thus, the maximum value of $T_i(k+1) - T_i(k)$ must occur when $i = 0$ and the minimum value must occur when $i = r(k) - 1$.

Thus if we can show that

$$r(k) \leq T_{r(k)-1}(k+1) - T_{r(k)-1}(k)$$

and that

$$T_0(k+1) - T_0(k) \leq r(k-1) \cdot (k \cdot (q(k) - 1) + 1)$$

for all k then we will be done. We will in fact show that we have equality in both cases; we will show that

$$T_{r(k)-1}(k+1) - T_{r(k)-1}(k) = r(k)$$

and

$$T_0(k+1) - T_0(k) = r(k-1) \cdot (k \cdot (q(k) - 1) + 1)$$

for all k.

By construction, the last entry of the schedule due to the left-most node at height k corresponds to $T_{r(k)-1}(k+1)$, and the last entry of the schedule due to the right-most child of that node corresponds to $T_{r(k)-1}(k)$. Since in a postorder depth-first traversal each node is visited immediately after its right-most child, we have that

$$T_{r(k)-1}(k+1) - T_{r(k)-1}(k) = r(k),$$

corresponding to the $r(k)$ entries of the schedule due to the left-most node at height k.

Meanwhile, $T_0(k+1)$ and $T_0(k)$ refer to the first entry of the schedule due to the left-most nodes at heights k and $k-1$ respectively. Thus $T_0(k+1) - T_0(k)$ will be equal the number of entries in the schedule due to the left-most node at height $k-1$, plus the number of entries due to all the subtrees whose roots are siblings of the left-most node at height $k-1$.

The first quantity is $r(k-1)$, by construction. As for the second quantity, since the subtrees in question each correspond to k occurrences of $r(k-1)$ units, and there are $q(k)-1$ such subtrees, we have that the second quantity is equal to $k \cdot r(k-1) \cdot (q(k)-1)$. Thus

$$\begin{aligned} T_0(k+1) - T_0(k) &= r(k-1) + k \cdot r(k-1) \cdot (q(k)-1) \\ &= r(k-1)q(k) \cdot k + r(k-1) - k \cdot r(k-1) \\ &= r(k) \cdot k - r(k-1) \cdot (k-1). \end{aligned}$$

Corollary 2 (Window Growth) *The minimal window length required for the generalized recap schedule, $b_k - a_k$, grows as $\Theta(k \cdot r(k))$*

This follows from the bounds above, since

$$
\begin{aligned}
b_k - a_k &= r(k) \cdot k - r(k-1) \cdot (k-1) - r(k) \\
&= r(k) \cdot (k-1) - r(k-1) \cdot (k-1) \\
&= (r(k) - r(k-1)) \cdot (k-1) \\
&= (r(k) - \frac{r(k)}{q(k)}) \cdot (k-1) \\
&= (1 - \frac{1}{q(k)}) \cdot r(k) \cdot (k-1)
\end{aligned}
$$

This calculation shows that $b_k - a_k$ must between $\frac{1}{2} r(k) \cdot (k-1)$ and $r(k) \cdot (k-1)$, since $q(k)$ must be a positive integer greater or equal to 2.

3.2 The Slow Flashcard Schedule

Here we examine the slow flashcard system in detail. In particular, we show that the slow flashcard schedule adheres to the spacing constraints

$$
\begin{aligned}
a_k &= k \\
b_k &= k^2.
\end{aligned}
$$

We also present evidence which suggests that the slow flashcard schedule even adheres to the more stringent constraints

$$
\begin{aligned}
a_k &= k \\
b_k &= 2k.
\end{aligned}
$$

We also show that for the slow flashcard schedule, t_n is bounded below by $\Omega(n^2)$ and bounded above by $O(n^3)$, and we present evidence that in fact t_n grows as $\Theta(n^2)$

We begin by re-examining the construction. We consider an infinite deck of flashcards, indexed by positions $1, 2, 3, \ldots$. We call position 1 the *top* or the *front* of the deck, we say that a flashcard in position i is *behind* another flashcard in position j if and only if $i > j$. Otherwise, it is *in front* of the other flashcard. Each flashcard corresponds to an educational unit u_i, and at the beginning of the construction, flashcard u_1 is in position 1, flashcard u_2 is in position 2, etc.

We construct the schedule as follows. At a given time step t, suppose that the flashcard at the top of deck corresponds to educational unit u_i, and that u_i has appeared in the sequence $k - 1$ times so far. Then we include u_i in the sequence at time step t (resulting in its kth occurrence), and we move the flashcard containing u_i to position $k + 1$ in the deck of flash cards.

Thus the configurations of the deck in the first few time steps are as follows:

$$u_1, u_2, u_3, u_4, u_5, u_6, \ldots$$

$$u_2, u_1, u_3, u_4, u_5, u_6, \ldots$$

$$u_1, u_2, u_3, u_4, u_5, u_6, \ldots$$

$$u_2, u_3, u_1, u_4, u_5, u_6, \ldots$$

$$u_3, u_1, u_2, u_4, u_5, u_6, \ldots$$

$$u_1, u_3, u_2, u_4, u_5, u_6, \ldots$$

$$u_3, u_2, u_4, u_1, u_5, u_6, \ldots$$

$$u_2, u_4, u_3, u_1, u_5, u_6, \ldots$$

$$u_4, u_3, u_1, u_2, u_5, u_6, \ldots$$

$$u_3, u_4, u_1, u_2, u_5, u_6, \ldots$$

resulting in the schedule

$$u_1, u_2, u_1, u_2, u_3, u_1, u_3, u_2, u_4, u_3, \ldots,$$

which simply corresponds to the units at the top of the deck (the left entries in the sequences above) at each time step.

As in the last section, we let $T_i(k)$ be the time step of the k^{th} occurrence of unit u_i in the schedule. Thus, for example, here we have that $T_2(3) = 8$, $T_4(1) = 9$, and $T_3(3) = 10$.

Note that, by construction, at every time step each flashcard except for the one being reinserted either maintains its position or moves up in the deck, decreasing its position by 1. The former happens if the presented flashcard is reinserted in front of the flashcard in question, and the latter happens if the presented flashcard is reinserted behind the flashcard in question. We call this the *slow marching property*, since informally it says that once a flashcard is inserted into position n, it will "slowly march" to the front of the deck, moving up at a rate of at most 1 position per time step.

Note also that if u_b is behind u_a in the deck at time step t then u_b will also be behind u_a at time step $t + 1$, unless u_a is in position 1 at time step t, and is reinserted behind u_b at the end of time step t. We call this the *no-passing property*.

Now we move on to proving the main results of this section.

Theorem 7 (Adherence to Spacing Constraints) *The slow flashcard schedule adheres to the spacing constraints*

$$\begin{aligned} a_k &= k \\ b_k &= k^2. \end{aligned}$$

To prove the theorem we need to show that

$$n \leq T_i(n+1) - T_i(n) \leq n^2$$

for all i and n. The left inequality follows from the slow marching property as follows. At $T_i(n)$ the flashcard u_i has been reinserted into position $n+1$ of the deck, by construction. Thus it will be at least n time steps until it is in position 1 by the slow marching property; it will be at least n time steps until $T_i(n+1)$. So

$$T_i(n+1) \geq T_i(n) + n$$

and from this we get the left inequality.

For the right inequality, again consider the time step $T_i(n)$, where u_i is presented for the n^{th} time and then removed from the deck and reinserted into position $n+1$. Immediately after $T_i(n)$, flashcard u_i is in position $n+1$. Because there is no passing, the part of the schedule in between $T_i(n)$ and $T_i(n+1)$ will consist only of the flashcards that are in front of u_i at $T_i(n)$.

Each time one of these flashcards is presented, it may be reinserted either in front of or behind u_i. Once it has been reinserted behind, it will not be shown again until at least $T_i(n+1)$, again by the no-passing property. Meanwhile, each time it is reinserted, it is reinserted further back in the deck than the previous time it was reinserted, by construction. Thus any flashcard can be presented/reinserted at most n times in between $T_i(n)$ and $T_i(n+1)$, one time for every position less than $n+1$ into which it could be reinserted. So in between $T_i(n)$ and $T_i(n+1)$, the possible reinsertions are limited to each of the n flashcards that are in positions $1, 2, \ldots, n$ at $T_i(n)$, each being reinserted at most n times. Thus

$$T_i(n+1) - T_i(n) \leq n^2.$$

In fact, since flashcards can't be reinserted into position 1, we have

$$T_i(n+1) - T_i(n) \leq n(n-1).$$

In any case, our proof is done.

Theorem 8 (Asymptotics of the Introduction Time Function) *In the slow flashcard schedule, $T_n(1)$ grows as $\Omega(n^2)$ and $T_n(1)$ grows as $O(n^3)$.*

We prove this by first showing that

$$T_1(n-1) < T_n(1) < T_1(n)$$

and then showing that $T_1(n)$ grows as $\Omega(n^2)$ and $T_1(n)$ grows as $O(n^3)$.

First, note that

$$T_1(n) < T_i(n)$$

for $i > 1$, for all n. Thus the first flashcard to be inserted into any given position will be the one corresponding to u_1. Thus for any n, flashcard u_n, which began in position n, will remain in position n until flashcard u_1 is reinserted into position n, at $T_1(n-1)$. Only after that can u_n make its way to the front of the deck and be presented for the first time. Thus,

$$T_1(n-1) < T_n(1).$$

At time $T_1(n-1)+1$, flashcard u_1 is right behind flashcard u_n. By the no-passing property, then, we get that

$$T_n(1) < T_1(n).$$

Thus we have that

$$T_1(n-1) < T_n(1) < T_1(n).$$

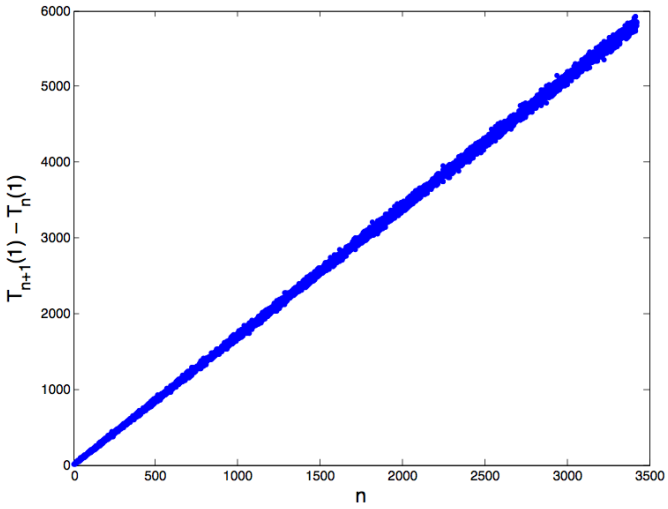

Figure 3.3: This figure shows $T_{n+1}(1) - T_n(1)$ plotted against n. The data is taken from the first 1,000,000 time steps of the slow flashcard schedule. A linear regression gives a line with slope 1.7, with a correlation coefficient of $r > .9997$.

Now note that, from the theorem above, we have that

$$n \leq T_i(n+1) - T_i(n) \leq n^2.$$

Thus $T_1(n+1) - T_1(n)$ grows as $\Omega(n)$ and as $O(n^2)$, and so $T_1(n)$ grows as $\Omega(n^2)$ and as $O(n^3)$.

We believe that both results above can be strengthened, and so we finish with two conjectures.

Conjecture 1 *For the slow flashcard schedule, $T_n(1)$ grows as $O(n^2)$, which would imply $T_n(1)$ grows as $\Theta(n^2)$.*

This conjecture is true if and only if $T_{n+1}(1) - T_n(1)$ grows as $O(n)$ and so as evidence for this conjecture we plot, in Figure 3, $T_{n+1}(1) - T_n(1)$ against n.

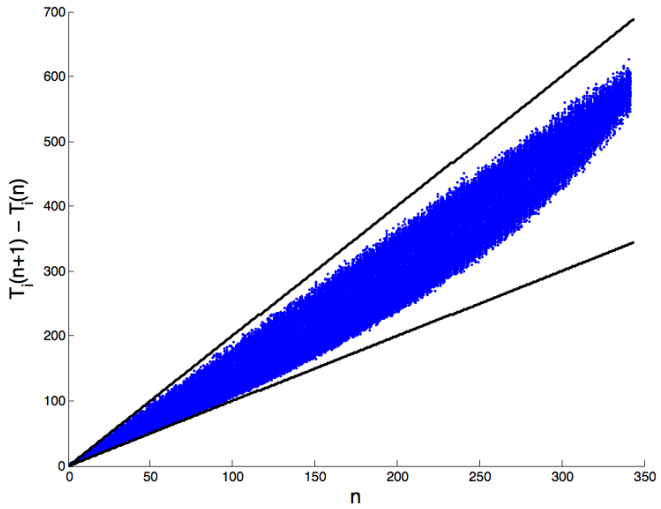

Figure 3.4: This figure shows $T_i(n+1) - T_i(n)$ plotted against n, for all i for which data was collected. The data is taken from the first 100,000 time steps of the slow flashcard schedule. Also shown are the lines going through the origin with slopes 1 and 2. All data points lie between the two lines.

Conjecture 2 *The slow flashcard schedule would exhibit infinite perfect learning with respect to spacing constraints with* $a_k = k$ *and* $b_k = 2k$.

This would be true if and only if

$$n \leq T_i(n+1) - T_i(n) \leq 2n$$

for all i and n. So as evidence for this conjecture we plot, in Figure 4, $T_i(n+1) - T_i(n)$ for all i.

3.3 Cramming

Here we establish bounds on how much can be crammed in a limited amount of time. Assume that spacing constraints $\{a_k\}$ and $\{b_k\}$ are given, as well as a pos-

itive integer T, and suppose there is a cramming sequence of length T that exhibits bounded learning of order n with respect to the given spacing constraints. We will derive an upper bound on n.

By the definition of bounded learning of order n, (i) the sequence adheres to the spacing constraints, and (ii) the sequence contains at least n distinct educational units such that, if the unit occurs a total of k times in the sequence, then its last occurrence is within b_k positions of the end of the sequence. (To be clear, this is to be interpreted to mean that the last element in the sequence is defined to be 1 position from the end of the sequence, not 0.)

Assume, without loss of generality, that these n units are labeled in reverse order of their last occurrences in the sequence. Thus unit u_1 is the last unit to appear in the sequence. Unit u_2 occurs for the last time before unit u_1 occurs for the last time, and so u_2 occurs for the last time at time step $t = T - 1$ at the latest. In general, for each i, unit u_i must appear for the last time at time step $t \leq T - i + 1$ at the latest — that is, at least i time steps from the end of the sequence.

Let $m(i)$ denote the smallest number k such that $b_k \geq i$. Then for every i, unit u_i must occur at least $m(i)$ times in the sequence, since otherwise the sequence would not satisfy part (ii) of the definition of bounded learning.

Since each of the n units must occur at least $m(i)$ times in the sequence, where i represents the label of the educational unit, and since each time step can afford at most one occurrence of one educational unit, we have that

$$\sum_{i=1}^{n} m(i) \leq T.$$

This represents an upper bound on n, since n must be such that this inequality holds true. (Note that the function $m(i)$ depends implicitly on the numbers in

$\{b_k\}$.)

Now consider just unit u_n. When it occurs last, it is for at least the $m(n)^{\text{th}}$ time. Since the spacing constraints must have been adhered to with respect to u_n, it follows that the the $m(n)^{\text{th}}$ occurrence of u_n must occur after a minimum of

$$\sum_{j=1}^{m(n)-1} a_j$$

time steps. And since it can occur no later than n time steps from the end of the sequence (that is, at time step $t = T - n + 1$), we have another statement on the minimum possible length of the sequence. Namely,

$$\Big(\sum_{j=1}^{m(n)-1} a_j \Big) + n \leq T.$$

Thus we have two inequalities, each of which represents an upper bound on n. In the language of scheduling theory, the first inequality represents a "volume bound", assuring that there is enough time for every unit to be seen as many times as it needs to be seen, and the second inequality represents a "path bound", assuring that the sequence is long enough to allow for even the unit which requires the longest time from the first occurrence to the end of the sequence.

Together the bounds incorporate the spacing constraints as well as the given amount of time. Nevertheless, for a given set of spacing constraints and a given T, the actual maximal n (that is, the maximal n such that a sequence of length T can exhibit bounded learning of order n with respect to the given spacing constraints) could be lower than the lower of these two upper bounds. This is because the bounds do not address the actual construction of cramming sequences, which appears in general to be a difficult scheduling problem which

hinges on the particulars of the spacing constraints. How to design general and efficient algorithms for constructing sequences which provably maximize cramming, so to speak, remains an open problem.

CHAPTER 4

MEASURING THE EFFECTIVENESS OF EDUCATIONAL SOFTWARE

Even usage data from seemingly simple educational software can be hard to interpret. Here we explore some of the challenges by considering an idealized mathematical model of educational software and its usage data. We consider the problem of measuring the mastery of the user as a function of time. We present a principled approach that combines cognitive modeling, data modeling and Bayesian inference to estimate the mastery of the user. We derive this method in the context of an idealized model user, and show how it can be adapted to more realistic models of the users of the software. We also describe a framework for gaining insight into how the method will perform with usage data from real students using real educational software. Finally, we exemplify how the approach can be applied to real data by applying it to actual usage data from a real-world educational iPhone app.

4.1 Motivation

Developing methods for evaluating the effectiveness of educational software is an important goal, but it presents challenges. One approach is to take the usage data for a given user and try to plot something like a "mastery curve" representing how much educational content the user knew, or had mastered, at each point in time as they used the software.

One difficulty with this approach is that any given interaction with educational software typically measures the user's mastery of at most a small amount of the total educational content handled by the software; generally software

cannot constantly test the user's mastery of the entire corpus of content. Meanwhile, the user's mastery of any given content changes over time, increasing or decreasing with exposure to the content or lack thereof. These phenomena underly the characteristic challenges involved in interpreting educational software usage data in general, and that pertain to the task of generating mastery curves in particular.

In this chapter we address some of these challenges by considering an idealized model of educational software, with a correspondingly idealized model of educational software usage data. This allows us to develop a general method for constructing mastery curves based on usage data for educational software.

The method we present here begins with modeling the notion of mastery itself, and modeling the users of the software. With these models in hand, we develop a principled method for estimating the mastery of a user from the usage data generated by that user.

A challenge perhaps even greater than developing methods for evaluating the effectiveness of educational software is the task of evaluating the effectiveness of these methods themselves. How can one possibly know whether one has done a good job of quantifying the mastery of a user through time?

In order to gain insight into the effectiveness of the method presented here, we first consider the effectiveness of the method in estimating the mastery of simulated model users whose true mastery – unlike that of real users – can be definitively compared to estimates based on the usage data.

By considering how the method performs in estimating the mastery of various model users, we develop an empirical framework for considering how the

method, or similar ones that employ the same principles, would perform in estimating the mastery of real users using real educational software.

The method presented here can be applied wholesale to real educational software usage data, aligning with one of the goals of the educational software data mining community: developing methods and tools that can be applied to educational software usage data in general and not just the usage data for which a specific method was developed [**?**]. The method can also be tailored to specific software, taking advantage of the particulars of the given usage data. More generally, the principles established in the context of the idealized models in this chapter may be applied in a wide variety of settings for extracting meaning from educational software usage data.

Thus in this chapter we provide not only methods that can be used on real educational software usage data with varying degrees of customization, but also principles that are more broadly applicable to educational software data mining more generally.

4.2 A Model of Educational Software and Usage Data

We begin by introducing an idealized model of educational software and usage data. This way we can study the process without the confounding factors that typically accompany actual educational software usage data. This model, like all the models in this chapter, is not meant to be realistic, but rather to capture the fundamental features that lead to the challenges of interpreting educational software usage data and generating mastery curves from the data.

We model the educational content of the software as a set of educational units, u_1, u_2, $\ldots u_N$. The educational units could represent vocabulary words, historical events, mathematical techniques, or anything else that might be considered one of many similar units in the context of educational software. This model is closely related to the model educational process studied in [**?**].

At every time step the model software does two things. First, it tests the user on an educational unit, resulting in a score of *pass* (P) or *fail* (F). If the user fails the test, the software teaches or reviews this educational unit for the student so that, either way, in the moment immediately after the time step in question, the user is ready to pass a second test on u_i. The teaching phase could represent the teaching of a mathematical competency by video lesson, a simple reminder of the definition of a vocabulary word, etc.

Then the usage data is supremely simple: a sequence of educational units and corresponding test results. An example of a usage dataset would be

$$(u_1, F), (u_2, F), (u_1, P), (u_2, P), (u_3, F), (u_1, F), \ldots$$

Here unit u_1 was not known by the user in advance of time step $t = 1$, then successfully recalled at $t = 3$ (after having been learned at $t = 1$), but then forgotten by $t = 6$. We denote the maximum time step as T and the total number of educational units represented in the usage data as N.

Of course in reality educational content is hardly a set of equivalent and unrelated units, and software is hardly a sequence of equivalent steps that determine if a user has mastered an educational unit and then teaches or reviews the unit if the test is failed. However this simple model, though highly idealized, will allow us to explore the challenges of generating mastery curves from educational software usage data without the distractions of the messiness of real

data. The model captures the essential feature of educational software which leads to the fundamental challenges in generating mastery curves from the usage data: *that educational software both tests and affects the user's mastery of content through time, but in a piecemeal fashion.*

4.3 A Method Based on Models of Mastery and Forgetting

We approach the task of generating a mastery curve from educational software usage data in three phases: 1) modeling the notion of the true mastery of the user – the thing which is to be estimated from the usage data, 2) modeling how mastery is gained and lost in the user's mind, and 3) devising a method for estimating the user's true mastery based on these models and the given usage data.

We imagine a researcher faced with usage data of the form described in the previous section, and tasked with generating a mastery curve. We describe idealized models of mastery and forgetting which our researcher could use as a basis for devising a method for estimating the user's mastery. Then we describe a method, informed by these models of mastery and users, for estimating the mastery of the user based only on the usage data.

4.3.1 A Model of Mastery

Underlying the endeavor to estimate how much a user has mastered at any given time step is the assumption that there is something to be estimated: some unseen quantity which cannot be observed directly but which is a measure of

how much the user has actually mastered. There are many ways to model mastery. The idealized model described here is based on consideration of the counterfactual question: Were the software to have tested the user on educational unit u_i at time t, would the user have passed the test?

We imagine that the answer to this question does exists somewhere, if somewhere inaccessible to our researcher, and with this in mind (no pun intended) we define mastery by saying that the user had mastery of u_i at time t if and only if she could have passed a test on u_i at time t.

More formally, we assume the existence of a table, internal to the user, which we call the mastery table. We denote the entry in the n^{th} row and t^{th} column as $M(n, t)$. If the user would have passed the test on unit u_n at time t then $M(n, t) = 1$. Otherwise $M(n, t) = 0$.

Then we define the total mastery of the user at time t as the number of educational units of which the user had mastery at time t. More formally, the mastery of the user at time t is defined to be the sum of the entries in the t^{th} column of the mastery table.

This is truly an unobservable quantity from the researcher's point of view, based as it is in counterfactual thinking. It does conform to an intuitive sense, though, for what is meant by colloquial usage of the term mastery. As with all the models in this chapter, it is a highly idealized one, but it has the essential characteristics which make its estimation from usage data a challenging task: 1) *It is something that can be measured by the software for any one educational unit at any one time, but not for all units at once,* and 2) *the user can be said to have mastery of an educational unit at one time step but lose it the next.*

4.3.2 A Model of Forgetting

We imagine our researcher modeling the user's propensity to lose mastery – to forget – with four basic assumptions. The first three assumptions are really three sides to one larger simplifying assumption: while the user is using the software, she is only gaining mastery of educational units from exposure to the educational units in the software, and not from any external educational source. The assumptions are: 1) if the user passes the first test on a given educational unit, say at time t, then she would have passed the test at any time step prior to t; 2) if the user passes a test for an educational unit at time step t_2, and the latest time step prior to t_2 wherein she was exposed to this unit was time step t_1, then she would have passed the test at any time step in between t_1 and t_2; 3) if a user would have have failed a test on a unit at time t then, unless she was exposed to this unit at time t, she would have also failed the test at time $t + 1$. The method we will present could be adapted to different assumptions, but we will use these to demonstrate how assumptions of this character can be used along with the last assumption in order to estimate mastery.

The last assumption governs how the user actually forgets educational units as a function of the amount of time since the last occurrence of that unit. The assumption is that as the user is using the software, she forgets any given educational unit at any given time step – that is, she loses mastery of the unit before being tested at that time step, and not before any other time step since the last occurrence of the unit – with a probability $f(\tau)$, where τ is the number of time steps since the last occurrence of the given unit. As a result, the probability that the user recalls an educational unit τ time steps after last being exposed to the unit – absent any further information such as the user's later ability to pass a

test on the unit – is described by another function of τ alone, namely

$$p(\tau) = \prod_{t=1}^{\tau}(1 - f(\tau))).$$

For now we avoid specifying the form of the function $f(\tau)$, and thus implicitly avoid specifying the form of $p(\tau)$ as well. The assumption is merely that the stochastic process governing forgetting is such that the probability of the user having mastery of a unit at a given time depends solely on the number of time steps since the last exposure of the unit.

We use this simple assumption for now so that we can address how to use $p(\tau)$ in estimating mastery based on usage data, but the method we describe could easily be adapted to more general and realistic models of forgetting. For example, the method could easily be adapted to a model where the rate of forgetting depends on the number of previous exposures of the educational unit in question (as well as the number of time steps since the last exposure). Such a model would use families of functions $f_k(\tau)$ instead of simply $f(\tau)$, where k represents the number of previous exposures to the unit. We will return to this point later on, and even implement and examine a method based on such a model of forgetting.

For now we stress that the method described below uses the extremely simple stochastic model for forgetting, $f(\tau)$, in order to make the method easier to describe, and easier to analyze.

4.3.3 The Method

We refer to the above models of mastery and forgetting combined as the model user. In this section we describe a method for estimating the mastery of a real user based on this model of a user. The method can be adapted to more complex model users, but will be described in terms of this simple one in order to focus the discussion on the process of using a model student to guide the development of a mastery-estimation method, rather than on one specific method.

The method begins by establishing a table with the same dimensions as the mastery table and where, to begin with, each entry is blank. We call this new table the credit table, and denote the entry in the n^{th} row and t^{th} column as $C(n, t)$.

The outline of the method is this: We fill in values for cells in C with the analogous values in M when they can be observed directly in the usage data. When a value for a cell can be deduced from the first three assumptions of the model user, we assign this value to the cell in C. Then, for the remaining cells (i, t), we use our final assumption to assign a value to $C(i, t)$ which corresponds roughly to the probability, given the assumptions and the usage data, that the analogous cell in M contains a 1 and not a 0. Once all the entries in the credit table are filled in, we sum up the columns and use these numbers as estimates of the analogous sums for the mastery table.

To begin with we go through each row in the usage data, and if the t^{th} entry is (u_i, P) then we set $C(i, t) = 1$. If the t^{th} entry is (u_i, F) then we set $C(i, t) = 0$. These entries correspond to to the non-counterfactual entries in the mastery table, since for these entries (i, t) the user actually was tested on unit u_i at time t. Based on our definition of mastery, we know that the entries just filled into C

are equal to the analogous entries in M.

Next, on the basis of the first three assumptions about forgetting, we fill in all the entries prior to the first occurrence of each unit with 1s if the first test on the unit was passed, and with 0s if the first test was failed. We also fill in, with 1s, all the cells in gaps between successive occurrences of educational units where the test was passed at the latter occurrence of the unit.

The remaining entries to be filled in fall into two categories: entries (i, t) where t is later than the time step of the last occurrence of u_i in the usage data, and entries (i, t) where the user failed the test on educational unit u_i on its next occurrence after time step t.

For entries in the first category, let us suppose that we are considering entry (i, t) where the last occurrence of unit u_i in the usage data was at time step $t - \tau$. Then the ideal value to put into cell $C(i, t)$, from the researcher's point of view, would be $p(\tau)$ since this is the probability with which $M(i, t) = 1$ instead of $M(i, t) = 0$.

Our researcher is not given $p(\tau)$ explicitly though. Here the method calls abstractly for using the usage data to estimate $p(\tau)$ with some function $\hat{p}(\tau)$.

One way to do this would be to fit a function $\hat{p}(\tau)$ to the data. That is, consider all the gaps between successive occurrences of educational units in the usage data, and for each τ for which at least one gap of length τ is observed, calculate $p_{\text{obs}}(\tau)$: the number of observed gaps of length τ for which the test was passed at the occurrence at the end of the gap, divided by the total number of observed gaps of length τ. Then fit a parametrized function $\hat{p}(\tau)$ to $p_{\text{obs}}(\tau)$, and then use $\hat{p}(\tau)$ as an approximation of $p(\tau)$.

It would likely behoove the researcher to look at the usage data and tailor the choice of parametrized function to the particulars of the data. A reasonable default choice might be to fit a logistic function, since the logistic family includes functions $\hat{p}$ such that $\hat{p}(0) \approx 1$ and $\hat{p}(\tau) \to 0$ as $\tau \to \infty$, which matches the natural intuition for what form $\hat{p}(\tau)$ should take. (Immediately after being exposed to a unit, the user is very likely to have mastery of the unit. As time goes on, the probability of having retained that mastery fades.)

Another approach would be to model the underlying stochastic process generating the data instead of modeling the data itself. That is, assume that $f(\tau)$ belongs to some parametrized family of functions and find the parameter values which maximize the likelihood of observing the given usage data. Then from this particular $\hat{f}(\tau)$ calculate the related function $\hat{p}(\tau)$ either analytically or numerically.

In any case, with $\hat{p}(\tau)$ – the researcher's best estimate of $p(\tau)$ – obtained one way or another, we are ready to fill in more cells in the credit table. For each (i, t) where the last occurrence of unit u_i in the usage data was at time step $t - \tau$ we set $C(i, t) = \hat{p}(\tau)$.

Now the only cells that remain to be filled in are cells between two successive occurrences of an educational unit where the test at the latter occurrence was failed. Suppose we are considering entry (i, t) where the occurrence of u_i previous to t was at time t_1 and where the next occurrence was at time t_2. Let $\tau = t - t_1$ and let $\gamma = t_2 - t_1$, so that (i, t) corresponds to a cell that is τ steps into a gap of length γ. Naively one may be tempted to set $C(i, t) = \hat{p}(\tau)$ again, however this fails to use all the information at hand.

We know that the user will later fail unit u_i at the next occurrence, thus giving the student partial credit of $p(\tau)$ is in a sense too generous. We would be ignoring a sort of selection bias, since we know the user will actually fail the next test on u_i. Another way to see what is wrong is to imagine a situation where $\gamma = \tau + 1$, and $p(\tau)$ is close to 1. Then the user would be getting nearly full credit for having mastered u_i at time t, even though it is known that the user failed the test on u_i just 1 time step later, earning 0 credit at that time step. Intuitively, something seems wrong with assigning credit such that there would be jerky drops from high partial credit to zero credit in such situations. If the user is known to fail a test on a unit at a given time step, then the chances that the user had mastery of the unit one time step earlier should be relatively small, and so the partial credit assigned should be correspondingly small.

What is really called for is the use of $\hat{p}(\tau)$ to calculate the *conditional* probability that the user had mastery of a unit τ units after the previous occurrence of it, *given* that the user did not have mastery of it γ time steps after the previous occurrence. This is the amount of credit we will assign to cells in the credit table that represent entries that are τ time steps into a gap of length γ, and where the test on u_i at the end of the gap was failed.

To calculate this conditional probability, we begin by letting X be a random variable denoting the number of time steps that elapse after an occurrence of a unit before the user forgets that unit. Then, by definition, $Pr(X \geq \tau) = p(\tau)$. What we are interested in now is

$$Pr(X \geq \tau \,|\, X < \gamma).$$

We calculate this using Bayes' rule that

$$Pr(A \mid B) = \frac{Pr(A \text{ and } \mathrm{B})}{Pr(B)}.$$

Since $Pr(X \geq \gamma) = p(\gamma)$ we have that

$$Pr(X < \gamma) = 1 - p(\gamma).$$

Also, since $Pr(X \geq \tau) = p(\tau)$ and $Pr(X \geq \gamma) = p(\gamma)$, and since $\gamma \geq \tau$, we have that

$$Pr(X \geq \tau \text{ and } \mathrm{X} < \gamma) = \mathrm{p}(\tau) - \mathrm{p}(\gamma).$$

Thus

$$Pr(X \geq \tau \mid X < \gamma) = \frac{p(\tau) - p(\gamma)}{1 - p(\gamma)}.$$

This is the value we assign to cells that are τ steps into a gap of length γ, where the test at the end of the gap was failed.

Thus we have filled in all of the entries in the credit table C. The estimate for the the mastery of the user at time t is then the sum of the entries in the t^{th} column of C. Like this we generate the mastery curve from the usage data.

4.4 Evaluating the Effectiveness Mastery Estimation Methods

Here we develop a framework for gaining insight into how effectively the method and similar methods would perform with real usage data by considering their performance in the context of simulations where usage data is generated by a simulated model student instead of a real person. In such a simulation, we can explicitly compare the true mastery of the simulated model student with the estimated mastery produced by the method.

We begin be considering numerical experiments where the method is expected to perform very well; experiments where the usage data is generated by a simulation of the same model student on which the mastery-estimation method is based. These experiments will demonstrate the power of the method to estimate the sums of the columns in the mastery table, using only the relatively sparse usage data, under ideal circumstances. After considering the method's performance under these circumstances, we will consider its performance under less ideal circumstances to gain insight into how well one should expect the method to perform with real data.

In our first simulation, usage data was generated as follows. First, a schedule of educational units for the educational software was generated by randomly choosing educational units from $u_1, \ldots, u_N$, with replacement, using $N = 40$. Like this we generate a schedule of length $T = 1000$.

Next we simulate a model student being exposed to educational units according to this schedule. The forgetting of the model student is determined by a function $f(\tau)$ where f is a Gaussian with mean N and variance $\frac{N}{6}$, truncated at 0. That is, the probability that the user forgets any given educational unit τ time steps after any given exposure is determined by a random variable with distribution $f(\tau)$. As the simulation is run, the true mastery table, internal to the model user, is generated. In addition, the usage data is generated and stored for use with the method.

Then we apply the method above to this usage data, generating a credit table. In the course of applying the method we use a logistic regression to find $\hat{p}(\tau)$, which is nearly perfectly suited to usage data generated with $f(\tau)$ as described above. (The imperfection comes from the truncation of the Gaussian,

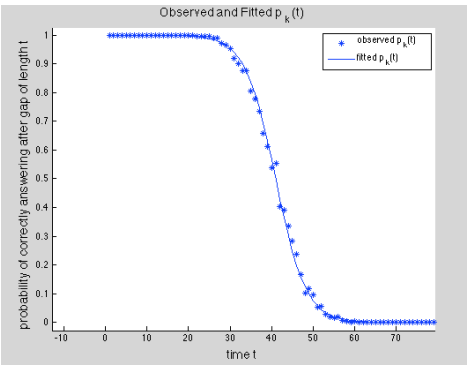

Figure 4.1: In the first numerical experiment, the fitted function $\hat{p}(\tau)$ is very good fit for the data $p_{\text{obs}}(\tau)$.

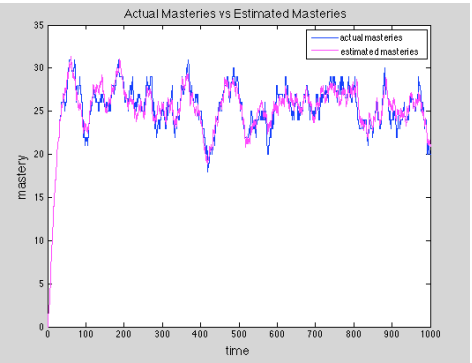

Figure 4.2: In the first numerical experiment, the estimated mastery is very close to the true mastery of the simulated model student.

but this truncation – of events six standard deviations from the mean – is relatively negligible.)

Figure 1 shows $p_{\text{obs}}(\tau)$ and $\hat{p}(\tau)$. As expected, the fit is relatively good. Figure 2 shows shows the estimated total mastery (the sums of the columns of the credit table) compared with the true total mastery of the model user (the sums of the columns of the mastery table). Here too the fit is, informally speaking, quite good. The estimated mastery catches much of the fluctuation of the true mastery. We do not attempt to quantify the goodness of the fit, although how to appropriately do so is another interesting question. Instead we stop at plotting

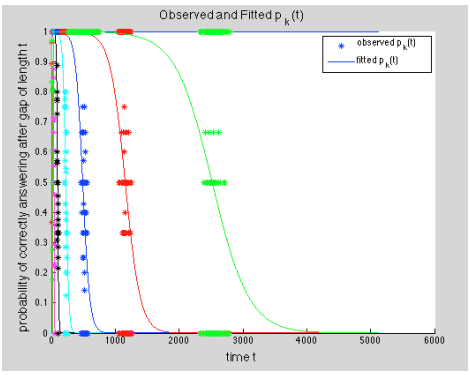

Figure 4.3: In the second numerical experiment, the functions $\hat{p}_k(\tau)$ are good fits for the data $p_{k,\text{obs}}(\tau)$. They are not as good as in the first experiment, in part because of the limited number of data points for each k.

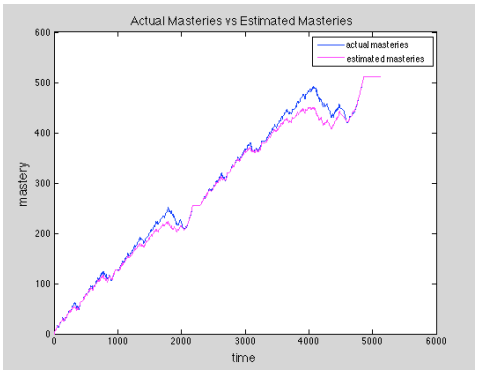

Figure 4.4: In the second numerical experiment, the estimated mastery is close to the true mastery of the simulated model student, even if the fit is not as good as in the first numerical experiment. The second experiment shows that even when the model user is relatively complex, mastery can still be well-estimated if the method is based on an appropriate model user.

the two curves and noting that, informally and qualitatively speaking, the fit is good.

Next we show the analogous figures for a different numerical experiment. Here again the method and the model user in the simulation are well suited to each other, but the model user is a bit more complex than the original model

user, and the method is correspondingly more complex as well.

In this experiment, we also make the schedule a bit more realistic; it is not simply random, as it is in the first experiment. Instead it is a schedule designed to be somewhat (though not perfectly) well suited to the same model user which will generate the usage data, and for which the method is designed. This is natural, as one would expect that the assumptions about the user that drive the design of the scheduling in the software would be similar to the assumptions driving the approach to analyzing the data; such would be the case if the designers of the software and the researchers looking at the usage data were in agreement about how students learn.

The schedule is essentially a randomized version of the recap schedule from [**?**]. In order to get the first $(k+1)\cdot 2^k$ entries in the schedule, start with an empty sequence, and a full binary tree of height k with nodes labeled $u_1, \ldots, u_{2^k}$, and then go through the tree using a post-order depth-first traversal. Upon visiting any leaf, append the sequence with the label of the leaf. Upon visiting a node, append a random permutation of the left descendent leaves of the node, and then a random permutation of the right descendent leaves of the node. This results in a sequence of length $(k+1)\cdot 2^k$ with exactly $(k+1)$ occurrences of each educational unit in $u_1, \ldots, u_{2^k}$. With this schedule, the gap between the k^{th} and $(k+1)^{\text{st}}$ occurrence of any unit will be between 2^{k-1} and $(k+3)2^{k-1}$.

The forgetting of the model student used in this simulation is determined by a family of functions $f_k(\tau)$ where f_k is a Gaussian with mean $\mu = (k+1)\cdot 2^{k-1}$ and variance $\sigma = \frac{\mu}{6}$. The probability that the user forgets a unit τ time steps after its k^{th} exposure is given by a random variable with distribution $f_k(\tau)$.

Thus this experiment models a student whose ability to recall an educational unit increases with repeated exposure, using educational software which schedules the educational units so that for any k, most but not all of the units should be recalled correctly (by this model student) upon the $(k+1)^{\text{st}}$ occurrence. This experiment is in a sense modeling a situation where software is somewhat but not perfectly well-suited to the user.

The method for generating the mastery curves used here is adapted to the same model student. In this case, for each k, data corresponding to gaps between the k^{th} and $(k+1)^{\text{st}}$ occurrences are grouped together to form $p_{k,\text{obs}}(\tau)$, in analogy to the earlier construction of $p_{\text{obs}}(\tau)$, and then modeled with $\hat{p}_k(\tau)$, in analogy to $\hat{p}(\tau)$ from the first experiment.

Then, to fill in gaps that are τ steps into gaps of length γ between the k^{th} and $(k+1)^{\text{st}}$ occurrences of a unit, where the latter occurrence resulted in a failed test, these functions $\hat{p}_k(\tau)$ are used to calculate the conditional probability of recalling a unit correctly τ time steps after the k^{th} occurrence given that it will later be answered incorrectly γ time steps after the k^{th} occurrence.

Figure 3 shows $p_{k,\text{obs}}(\tau)$ and the fitted curves $\hat{p}_k(\tau)$. The fit of the data points to the curve is not quite as obviously good as earlier, but this is due to the relative lack of data going into each data point for each k. Here, since there are only 256 educational units in the simulation, there are only 256 data points informing each curve $p_k(\tau)$, and these are divided among the various represented gaps τ for each k.

Figure 4 shows the estimated mastery as compared to the true mastery. Here, again, we see the method doing a relatively good job of estimating the true

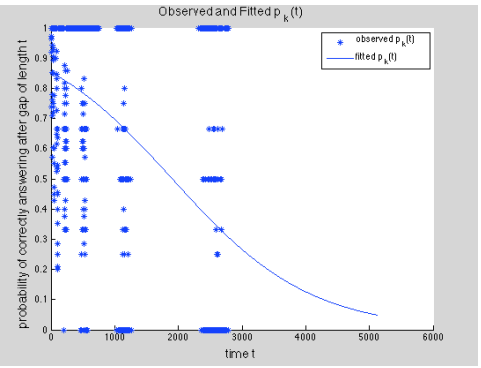

Figure 4.5: In the third numerical experiment, the fitted function $\hat{p}(\tau)$ is a lousy fit for the data $p_{\text{obs}}(\tau)$ because the model user generating the usage data does not match the model user informing the method. (In this numerical experiment, the model user from the first numerical experiment was used to fit data from the second numerical experiment.)

mastery of the model user.

These first two simulations show the method operating under ideal circumstances – where the model user generating the usage data matches the model user for which the method is tailored – and indeed performing well. The accuracy of the method in estimating mastery through time will depend on how well the model user informing the method models the actual user using the software.

To gain some insight into what happens when there is a mismatch, we consider a third experiment: we use the mastery estimation method from the first experiment, but we use the usage data from the second experiment. Here there is a gross mismatch between the method and the user: it's as if a researcher is estimating mastery based on the assumption the the user does not get better at retaining content through repeated exposure, but the user generating the usage data actually does indeed get better at it with repeated exposure (as measured here by the increasing means in the family of functions $f_k(\tau)$ as k increases.)

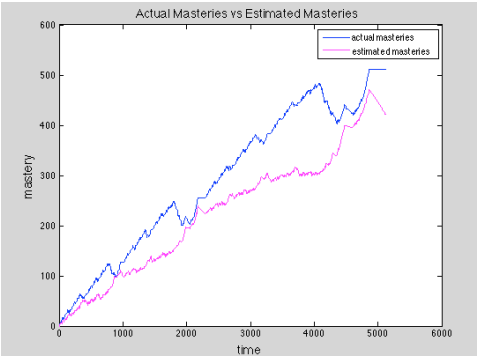

Figure 4.6: In the third numerical experiment, the estimated mastery is not close to the true mastery of the simulated model student, except for brief periods.

Figures 5 and 6 show the results of this experiment. Of course here $\hat{p}(\tau)$ does a relatively lousy job at explaining the usage data which it models, and this results in estimates of mastery that are off as well. Because much of the credit table is filled in without use of $\hat{p}(\tau)$ at all, the estimates are not as bad as they could be. (And the reason the estimates and the true mastery periodically become very close is that the schedule is such that it periodically contains brief runs which cycle through all the educational units which have hitherto appeared at least once. During these runs – analogous to the "recaps" referred to in [**?**] – the usage data contains information with which the method produces relatively good estimates of the mastery, mostly due to the first three assumptions of the model user and less reliant on $\hat{p}(\tau)$ than other periods in time. Other noticeable details of this figure can be explained by careful analysis of the schedule, but as such analysis would not shed significant light on the method, we omit it here.

4.5 Applying the Method to Real World Data

The general approach of the method described in this chapter is to make a parametrized model of the user that generated the usage data, fit the model to the data, and then use this to calculate the conditional probability that the user had mastery of a given item when it cannot be deduced from other assumptions about the user or simply read off from the usage data directly. The approach can work well, but is limited by statistical differences between the actual user and the model user. With real world data, the challenge will be to use a model which matches the real users of the software.

That said, the are many tools available for constructing appropriate model users. Above we describe a family of functions $p_k(\tau)$, one for each number of prior exposures to the unit, but one could do much more than that.

For example, if usage data for a population of users were available, one could use the population-wide data to calculate a different $\hat{p}_k(\tau)$ for each educational unit – call it $\hat{p}_{k,u_i}(\tau)$ – and use this function instead of the general $\hat{p}_k(\tau)$ based on the individual user's data. (One could not fit functions $\hat{p}_{k,u_i}(\tau)$ based only on the individual user's data, since there would be only one data point per user on which to base such a function.) This approach would take into account differences in the difficulties of educational units, implicitly extending the idealized model of educational software presented in this chapter.

Even better, one could use the population-wide data to see how the forgetting rates typically compare between educational units, and then use this information to adjust the parameters of the individual-based $\hat{p}_k(\tau)$ according to which educational unit is in question. This would thus use a hybrid of indi-

vidual data and population-wide data. The function $\hat{p}_k(\tau)$ would originally be based on the usage data of the individual, aggregated over all units, but would then be adjusted for each unit according to the known differences in the difficulties of the units.

Even starting with the principles and frameworks presented in this chapter, customizing the method for real usage data is a formidable task – one far beyond the scope of this chapter, or this dissertation. We finish by presenting a naive application of the method, more for the sake of demonstration than extracting useful insights into the usage data being examined.

In Figure 7, we see the results of applying the method to some usage data from Flash of Genius: SAT Vocab. The data used is of the same form as described above, since this application is one where at every time step the user sees a vocabulary word and reports (via tapping an "X" or a check mark) whether they could recall the definition of the word before seeing the definition. Naturally there is no "actual mastery" curve here, since this is a real user and not a simulated one as in our numerical experiments.

4.6 Summary

The purpose of this chapter is to present idealized models of educational software, usage data and mastery through which we can cleanly explore the challenges at hand; to show how to use models of users to estimate mastery in the context of these idealized models; and to demonstrate the simulation-based framework for gaining insight into how effectively the mastery-based estimation should be expected to perform under various circumstances. We hope that

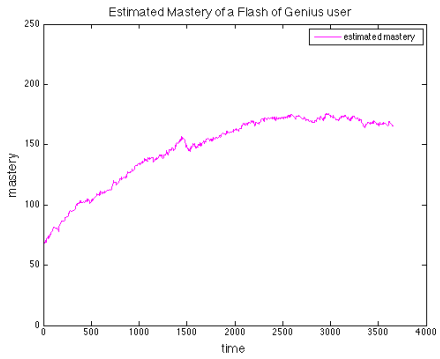

Figure 4.7: Here we see the results of the method of the second numerical experiment applied to the usage data of a real user of Flash of Genius: SAT Vocab. According to these estimates, the user already knew 66 of the vocabulary words presented before using the application, and gained mastery of about 120 words over the course of about 3500 time steps, each representing the viewing of a flashcard, sometimes losing mastery along the way.

the models and frameworks presented here can serve as a foundation for more in-depth explorations into methods for generating mastery curves from educational software usage data, and that the models and frameworks more generally contribute to a growing body of work which can serve as a foundation for intuition for researchers and engineers working in the growing fields and industries of educational software data mining.

CHAPTER 5

THE CONCEPT REINFORCEMENT PROBLEM

If vocabulary words are more easily learned when etymologically related words are already known, how might an educational software designer find the optimal order in which to teach a set of vocabulary words? Motivated by this problem, and more broadly by the goal of developing a theoretical foundation for an algorithmic theory of education, we introduce a novel networks-related optimization problem. The goal is to minimize the cost of sequentially installing the nodes of a given network, where the cost of installing a given node is a function f of the number of neighbors of the given node that have already been installed when the given node is being installed. The case of convex and decreasing f is shown to have structure which suggests certain heuristic approaches to the problem. The problem is shown to be trivial when f is affine linear. The general problem where f is allowed to be increasing is shown to be NP-hard.

5.1 Motivation

An educational software designer working on a vocabulary-building application may seek the optimal order in which to present a set of vocabulary words to the user, under the assumption that a word like *neologism*, for example, would be easier for students to learn if they already knew the etymologically related words *neophyte* and *epilogue*.

More generally, cognitive psychologists and researchers who study learning in humans may seek to quantitatively understand how familiarity with some concepts affects one's ability to learn others. Such research could in turn inspire

progress in the fields of artificial intelligence and machine learning.

The notion that the cost of installing a node in a network could depend on whether neighboring nodes have already been installed could find other applications as well. For example, after a natural disaster affecting a coastal region, the cost of rehabilitating inland facilities could depend on whether or not a nearby port had already been rehabilitated, while conversely the cost of rehabilitating the port could depend on whether nearby inland facilities were available to support the effort. The optimal course of action may be to reconstruct inland facilities and ports in a carefully chosen order.

Despite a variety of potential applications, this simple notion has not received attention in the literature. There has been an enormous amount of research published on the Traveling Salesman Problem, the related problem where the cost associated with a given node is a function of the given node and *the previously visited node*. But seemingly absent from the literature is consideration of any problem where the cost associated with a given node is a function of the given node and *the entire set of adjacent nodes that will be visited before the given node*. In particular, the case where the function is a decreasing and convex function of the *number* of previously-visited adjacent nodes is natural to consider. It models a situation where there are efficiencies to be gained by having more nearby nodes already installed (or visited) when installing a given node, and where there are diminishing returns on those efficiencies.

With this in mind, we introduce the *Concept Reinforcement Problem*. We do not investigate the problem in detail here, giving only the most preliminary results. We show only that the problem is NP-hard for general f, and that it is trivial if f is constrained to be affine-linear. For the case of convex and decreasing f, we

show that the problem has structure that suggests specific heuristic approaches to the problem.

5.2 The Problem

Formally, the concept reinforcement problem is as follows. A network $G = (V, E)$ is given, along with a real-valued function $f(k)$, defined over the non-negative integers. The goal is to find the permutation, σ, of the nodes that minimizes the total cost of installing all the nodes in the network. The cost of installing node v_i in G under the permutation σ is given by

$$c_G^\sigma(v_i) = f(k)$$

where $k(i, G, \sigma)$ is the number of nodes adjacent to v_i in G that appear before v_i in the permutation σ. The total cost of installing G according to the permutation σ is given by

$$C_G(\sigma) = \sum_{i=1}^{n} c_G^\sigma(v_i)$$

where $n = |V|$.

When f is constrained to be decreasing ($i < j$ implies $f(i) > f(j)$) and convex ($i < j$ implies $f(i) - f(i+1) > f(j) - f(j+1)$) we call the problem the *concept reinforcement problem*, inspired by the original motivation for the problem.

5.3 Results

We do not present any deep theoretical results for either problem here, limiting ourselves for now to only some cursory observations.

5.3.1 The Concept Reinforcement Problem is NP-hard

The concept reinforcement problem is NP-hard. This can be established using a quick reduction from the NP-hard problem Maximum Independent Set. Given a graph G, we can find a maximum independent set by seeking the optimal solution to the instance of our problem with the same G and the function f defined by $f(0) = 0$ and $f(k) = 1$ for all $k > 0$. Using this f, the optimal permutation found by solving our problem will necessarily have the maximum possible number of nodes with cost 0, and the rest with cost 1. The nodes with cost 0 then necessarily form a maximal independent set, since if any two were adjacent then the latter of the two to be installed would have incurred a cost of $f(1) = 1$.

5.3.2 The Concept Reinforcement Problem with Affine-Linear f is Trivial

Here we show that the cost of installing a network is independent of the order in which the nodes are installed if f is affine linear. More precisely, if $f(k) = ak + b$ then

$$C_G(\sigma) = am + bn$$

where $m = |E|$ is the number of edges in G, and $n = |V|$ is the number of nodes. The calculation is straightforward. Letting $k_i(\sigma)$ be the number of nodes adjacent to the i^{th} node in σ and appearing before that node in σ, we have that

$$
\begin{aligned}
C_G(\sigma) &= \sum_{i=1}^{n} f(k_i(\sigma)) && (5.1)\\
&= \sum_{i=1}^{n} (ak_i(\sigma) + b) && (5.2)\\
&= \left(a \sum_{i=1}^{n} k_i(\sigma)\right) + bn && (5.3)\\
&= am + bn. && (5.4)
\end{aligned}
$$

The fact that

$$\sum_{i=1}^{n} k_i(\sigma) = m$$

can be seen to be true by noting that each edge in G contributes exactly 1 to exactly one k_i in the following sense: if node u_1 appears before node u_2 in σ, and u_2 appears in position j, then edge (u_1, u_2) contributes 1 to $k_j(\sigma)$. All the edges are accounted for in this manner.

5.3.3 The Concept Reinforcement Problem Affords Hill-Climbing

The concept reinforcement problem with f constrained to be decreasing and convex is perhaps the most natural case to consider. It models situations where there are efficiencies to be gained from having prior-installed neighbors at the time of installation for a given node, but where there are diminishing returns on these efficiencies. For this problem, there is a useful hill-climbing algorithm for navigating solution space.

The basic idea is that if two neighboring nodes are being installed consecutively according to a given permutation, and if the cost incurred by the earlier

one is $f(a)$ but the cost incurred by the latter one is $f(b)$ where $b \leq a - 2$, then a cheaper permutation would result from switching the order of these two nodes in the permutation. The reasoning is as follows. After switching, the costs become $f(a-1)$ and $f(b+1)$, since the switch results in the originally-earlier node losing a prior-installed neighbor, and the originally-latter node gaining a prior-installed neighbor. The new combined cost for these two nodes is lower than the original combined cost for these two nodes by convexity ($b \leq a - 2$ implies $f(a) + f(b) > f(a-1) + f(b+1)$), and the costs for all the other nodes remains the same because the two nodes were consecutive in the permutation.

5.4 Future Work

This chapter merely introduces the concept reinforcement problem and gives only the most preliminary sort of results for them. Very basic questions remain unanswered. Is the general concept reinforcement problem also NP-hard? If not, how can the optimal solution be found in polynomial time? If so, what specific heuristic approaches work the best? And how does that depend on f or the nature of the graph? Many avenues of inquiry that have been pursued for the traveling salesman problem have analogues with the two problems introduced here.

Research into this problems is in its infancy. It is not only of practical interest for designers of educational software, but of theoretical interest as well, as it represents an interesting variant of the famous traveling salesman problem which has curiously been overlooked until now.

5.5 Acknowledgements

Credit goes to Joel Nishimura and Joel Lewis for each and independently proving the NP-Hardness result in this chapter. Thanks to Sasha Gutfraind and Milan Bradonjic for work on this problem extending the results shown here.

CHAPTER 6

CONCLUSION

Humans make better teachers than computers do, and that is likely to be true for many years to come. There is only so much to be gained by scheduling educational content perfectly. There is much more to be gained by inspiring students in ways that only a human teacher is capable of doing.

Nevertheless, it is clear that software will come to pervade education in the near future. Software may not replace teachers, but it seems likely to replace most textbooks. The advent of tablet computers such as the iPad and the Kindle, and netbooks such as the One Laptop Per Child computer, likely signal a vastly diminishing role for paper textbooks in schools – even in third-world schools – and a corresponding increase in the use of educational software.

With educational software bound to play an increasingly important role in the near future, it is incumbent upon the scientific community to lay the foundations for truly adaptive educational software. An algorithmic theory of education would contribute to this effort.

The theme of this dissertation is idealized models around education. The models presented are very simple mathematical constructs, and yet – perhaps surprisingly – they have not shown up in the literature before. The reason these simple mathematical constructs have not been the focus of research before is that their structures are just specific enough to education that one should actually only expect them to come up when educational software becomes the focus of computer scientists and applied mathematicians.

Take, for example, the basic combinatorial question in Education of a Model Student. Given weakly increasing sequences $\{a_k\}$ and $\{b_k\}$, is it possible to construct a sequence such that the $(k+1)^{\text{st}}$ occurrence of any element in the sequence occurs between a_k and b_k places to the right of the k^{th} occurrence? This is a very simply-stated question and, in the author's experience, mathematicians are often intrigued by the question upon hearing it, delighted by its simplicity and motivation, and somewhat incredulous that it has not been studied before. And yet that appears to be the case.

Upon more careful consideration, however, it is not surprising that this problem has not been studied before: it's posing is justified by a very natural application to educational software, which has never been the focus of researchers in combinatorics, algorithms or applied mathematics before, and it is not particularly well justified by applications outside of educational software. The reason that it is so specific to educational software is that the spacing constraints are modeling constraints imposed by the way in which humans – and, as the psychology literature tells us, other mammals – learn. They are modeling a natural phenomenon which is based on the neurobiology of learning. Few if any important physical systems or industrial applications have constraints that are well modeled by these spacing constraints. (The closest application the author is aware of is the scheduling of information transmission from multiple satellites to a single ground station that motivated research into the Pinwheel Scheduling Problem [**?**].)

Similarly, as pointed out explicitly in Chapter 4, the analysis of usage data from educational software has various characteristics – again all stemming ultimately from the biology of learning and forgetting – that are specific to educa-

tional software usage data rather than software usage data more generally.

So too with the concept reinforcement problem. This is a very simply-stated yet nontrivial networks problem that is so closely related to Traveling Salesman Problem that it is almost hard to believe that it has evaded investigation until now. Researchers hoping to burnish their CVs by working on a novel networks problem lament how difficult it is to come up with interesting, simply-stated yet un-investigated problems in their field. But the issue is likely not the lack of potentially interesting, simply-stated, un-investigated problems in network theory. Rather the issue is likely the continued focus on transportation and infrastructure – applications which have indeed been plumbed for years now – as a source of inspiration for novel network problems, instead of attempting to solve real problems coming out of new areas of science or industry.

The concept reinforcement problem was inspired by the author's work on an iPhone app for studying vocabulary words aided by their Latin and Greek roots. When it was time to decide in what order the words should appear in the application, it felt impossible for the author *not* to formulate the network optimization problem. Likewise, all the problems in this dissertation stemmed from efforts to smartly design the algorithms in this iPhone application, or to effectively analyze the usage data. The problems considered herein were all originally complicated by particulars of the specific software being programmed at the time, but stripped down to bare essentials – reduced to mathematical structures – before analysis so that they could be understood in terms of educational software in general, instead of only in terms of Flash of Genius: SAT Vocab.

It is the author's hope that in the future, software engineers writing educational software will profit from some of the intuitions that come from reading

the analysis of the problems presented here. Indeed, at the time the author was designing the application (in 2008), he wished there had been algorithmic education theory literature from which to gain valuable intuitions.

What this dissertation represents is of course not nearly enough. It represents just a tiny step towards an algorithmic theory of education, and one that exists in the context of other work can be seen as working toward this end as well.

For the applied mathematics and computer science community, there is an imperative to lay the foundations for adaptive educational software. This includes work on simple, idealized models such as those presented here, but also more detail-oriented modeling and investigations into algorithmic aspects of educational software not touched upon in this dissertation. One could only speculate about what sustained focus on building an algorithmic theory of education will bare.

It is an exciting time to be an applied mathematician or computer scientist with an interest in education, as the burgeoning educational software industry provides fodder for intellectually stimulating work that will lay the foundations for engineering that is likely to have a profound impact on human progress.

BIBLIOGRAPHY

[1] Novikoff T, Kleinberg J, Strogatz S (2012) Education of a Model Student. Proceedings of the National Academy of Sciences 109:1868-1873.

[2] Obama B (2009) Remarks by the President at the National Academy of Sciences annual meeting http://www.whitehouse.gov/the_press_office/Remarks-by-the-President-at-the-National-Academy-of-Sciences-Annual-Meeting/.

[3] Ebbinghaus H (1885) *Memory: a Contribution to Experimental Psychology*, translated by Ruger H, Bussenius C (1913) from German. (Teachers College at Columbia University, New York).

[4] Dempster F (1988) The spacing effect: a case study in the failure to apply the results of psychological research. American Psychologist 43:627–634.

[5] Balota DA, Ducheck JM, Logan JM (2007) in Foundation of Remembering: Essays in Honor of Henry L. Roediger III, ed Nairne, JS (Psychology Press, New York), pp 83–105.

[6] Cepeda NJ, Pashler H, Vul E, Wixted JT, Rohrer D (2006) Distributed practice in verbal recall tasks: a review and quantitative synthesis. Psychological Bulletin 132:354–380.

[7] Crowder RG (1976) Principles of Learning and Memory. (Psychology Press, New York).

[8] Roediger HI, III, Karpicke JD (2011) in Successful Remembering and Successful Forgetting: A Festschrift in Honor of Robert A. Bjork, ed Benjamin AS (Psychology Press, New York), pp 23–48.

[9] Melton AW (1970) Situation with respect to spacing of repetitions and memory. Journal of Verbal Learning and Verbal Behavior 9:596-606.

[10] Bahrick HP, Bahrick LE, Bahrick AS, Bahrick AP (1993) Maintenance of foreign language vocabulary and the spacing effect. Psychological Science 4:316–321.

[11] Landauer T, Bjork R (1978) in Practical Aspects of Memory. (Academic Press, New York), pp 625–632.

[12] Pimsleur P (1967) A memory schedule. The Modern Language Journal 51:73–75.

[13] Wozniak PA, Gorzelanczyk EJ (1994) Optimization of repetition spacing in the practice of learning. Acta Neurobiologiae Experimentalis 54:59-62.

[14] Wolf G (2005) Want to remember everything you'll ever learn? Surrender to this algorithm. Wired Magazine, Issue 16.05.

[15] Hyun E, Kim S, Jang S, Park S (2008) Comparative study of effects of language instruction program using intelligence robot and multimedia on linguistic ability of young children. Robot and Human Interactive Communication pp 187–192.

[16] Romero C (2010) Educational Data Mining: A Review of the State of the Art. IEEE Transactions on Systems, Man, and Cybernetics 40:601–618

[17] Holte R, Mok A, Rosier L, Tulchinsky I, Varvel D (1989) The pinwheel: a real-time scheduling problem. Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences 2:693-702